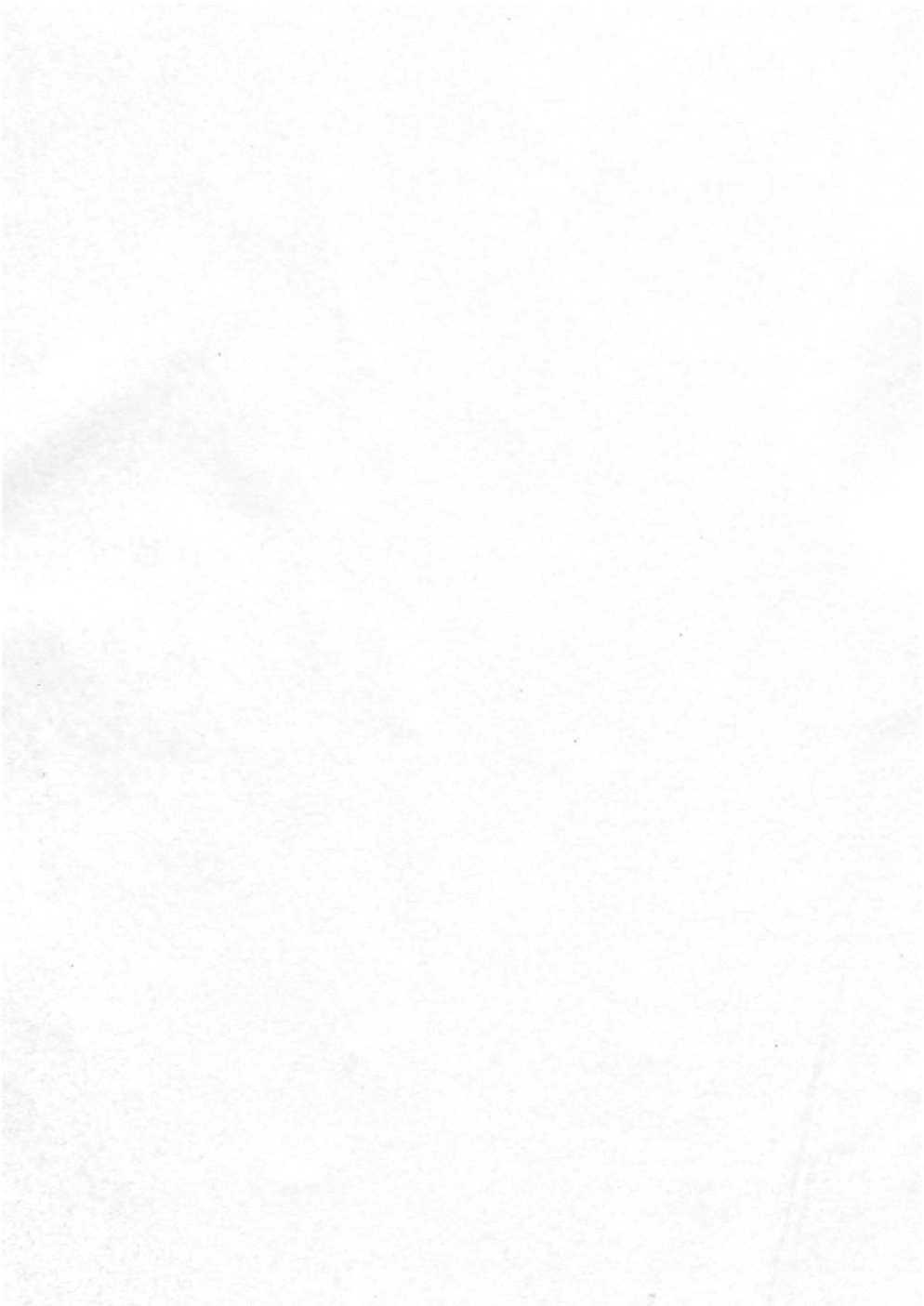
Das Programmier-Handbuch der IBM /370

Germain

Ein Lehr- und Arbeitsbuch

Carl Hanser Verlag
München Wien

Das Programmier-Handbuch der IBM /370



Clarence B. Germain

Das Programmier-Handbuch der IBM /370

Ein Lehr- und Arbeitsbuch

mit 401 Abbildungen, 64 Aufgabengruppen
und 14 Anhängen

Neufassung des Werkes
„Das Programmier-Handbuch der IBM/360"
vom Verfasser bearbeitet und stark erweitert.

Übersetzt von Rüdiger Gritsch

Carl Hanser Verlag München Wien 1976

Gesellschaft für Mathematik und
Datenverarbeitung mbH Bonn
— Bibliothek —
Inventar-Nr. 21352

Neufassung des Werkes "Das Programmier-Handbuch der IBM /360",
vom Verfasser bearbeitet und stark erweitert.

Titel der amerikanischen Originalausgabe
"Programming the IBM 360"
1967 by Prentice-Hall, Inc., Englewood Cliffs, N.J. USA

Clarence B. Germain ist Senior Systems Analyst
bei Computype, Inc., St. Paul, Minnesota

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Germain , Clarence B.
Das Programmier-Handbuch der IBM/370 IBM dreihundertsiebzig : e. Lehr- u. Arbeitsbuch. -
5. Aufl., Neufassung d. Werkes Das Programmier-Handbuch der IBM/360 / vom Verf. bearb. u. stark erw.
Einheitssacht.: Programming the IBM three hundred and sixty dt.
ISBN 3-446-11896-9

1.-4. Auflage
1969 - 1972 Das Programmier-Handbuch der IBM /360
5. Auflage Das Programmier-Handbuch der IBM /370

Alle Rechte vorbehalten
© 1976 Carl Hanser Verlag München Wien
Druck: Friedrich Pustet, Regensburg
Printed in Germany

Vorwort

Dieses Buch ist als Arbeitsunterlage für denjenigen gedacht, der die Programmierung des Datenverarbeitungssystems IBM /370 erlernen will oder muß. Es kann sowohl für den Klassenunterricht als auch für das Selbststudium herangezogen werden.

Es werden keine Vorkenntnisse vorausgesetzt, nicht einmal Vorkenntnisse über Lochkarten. Abgesehen von den wenigen in diesem Buch aufgeworfenen Problemen werden auch keine Mathematikkenntnisse vom Leser erwartet, die über das Oberschulniveau hinausgehen. Sowohl kommerzielle als auch wissenschaftliche Anwendungen werden erwähnt; die Aufgaben entstammen beiden Problemkreisen. Lösungen sind für rd. 150 Aufgaben präpariert.

Die Themen werden in der Reihenfolge behandelt, in der sie öfters besprochen werden. So weit das praktikabel erschien, sind die Kapitel über PL/1, FORTRAN, COBOL und die AL voneinander unabhängig, so daß der Leser die Themen auslassen kann, mit denen er sich nicht beschäftigen muß.

Das Buch berücksichtigt die Modelle des Systems IBM /370 mit virtuellem Speicher. Mit Ausnahme einiger Abschnitte (über VSAM und andere) treffen die Ausführungen für alle Modelle der IBM /370 zu.

Im allgemeinen werden das Betriebssystem DOS und das Betriebssystem OS in gesonderten Kapiteln behandelt, so daß der Leser diejenigen übergehen kann, mit denen er sich aufgrund seiner beruflichen Tätigkeit nicht beschäftigen muß. Das CMS (Conversational Monitor System) wird nicht behandelt.

Es ist das Ziel dieses Buches, den Leser so weit zu fördern, daß er die IBM-Literatur mit Gewinn studieren kann. Deshalb wurde auch nicht versucht, die besonderen Sprachelemente der einzelnen Programmiersprachen dem Leser nahezubringen.

Neben dem Buch sollte sich der Leser zum Studium auch eine Zeichenschablone und geeignete Programmschemen beschaffen. IBM-Handbücher über das System IBM /370 in einer angemessenen Zahl können ihm ebenfalls beim Studium behilflich sein.

St. Paul, Minnesota

Clarence B. Germain

Neufassung des Werkes "Das Programmier-Handbuch der IBM /360",
vom Verfasser bearbeitet und stark erweitert.

Titel der amerikanischen Originalausgabe
"Programming the IBM 360"
1967 by Prentice-Hall, Inc., Englewood Cliffs, N.J. USA

Clarence B. Germain ist Senior Systems Analyst
bei Computype, Inc., St. Paul, Minnesota

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Germain , Clarence B.
Das Programmier-Handbuch der IBM/370 IBM dreihundertsiebzig : e. Lehr- u. Arbeitsbuch. -
5. Aufl., Neufassung d. Werkes Das Programmier-Handbuch der IBM/360 / vom Verf. bearb. u. stark erw.
Einheitssacht.: Programming the IBM three hundred and sixty dt.
ISBN 3-446-11896-9

1.-4. Auflage
1969 - 1972 Das Programmier-Handbuch der IBM /360
5. Auflage Das Programmier-Handbuch der IBM /370

Alle Rechte vorbehalten
© 1976 Carl Hanser Verlag München Wien
Druck: Friedrich Pustet, Regensburg
Printed in Germany

Vorwort

Dieses Buch ist als Arbeitsunterlage für denjenigen gedacht, der die Programmierung des Datenverarbeitungssystems IBM /370 erlernen will oder muß. Es kann sowohl für den Klassenunterricht als auch für das Selbststudium herangezogen werden.

Es werden keine Vorkenntnisse vorausgesetzt, nicht einmal Vorkenntnisse über Lochkarten. Abgesehen von den wenigen in diesem Buch aufgeworfenen Problemen werden auch keine Mathematikkenntnisse vom Leser erwartet, die über das Oberschulniveau hinausgehen. Sowohl kommerzielle als auch wissenschaftliche Anwendungen werden erwähnt; die Aufgaben entstammen beiden Problemkreisen. Lösungen sind für rd. 150 Aufgaben präpariert.

Die Themen werden in der Reihenfolge behandelt, in der sie öfters besprochen werden. So weit das praktikabel erschien, sind die Kapitel über PL/1, FORTRAN, COBOL und die AL voneinander unabhängig, so daß der Leser die Themen auslassen kann, mit denen er sich nicht beschäftigen muß.

Das Buch berücksichtigt die Modelle des Systems IBM /370 mit virtuellem Speicher. Mit Ausnahme einiger Abschnitte (über VSAM und andere) treffen die Ausführungen für alle Modelle der IBM /370 zu.

Im allgemeinen werden das Betriebssystem DOS und das Betriebssystem OS in gesonderten Kapiteln behandelt, so daß der Leser diejenigen übergehen kann, mit denen er sich aufgrund seiner beruflichen Tätigkeit nicht beschäftigen muß. Das CMS (Conversational Monitor System) wird nicht behandelt.

Es ist das Ziel dieses Buches, den Leser so weit zu fördern, daß er die IBM-Literatur mit Gewinn studieren kann. Deshalb wurde auch nicht versucht, die besonderen Sprachelemente der einzelnen Programmiersprachen dem Leser nahezubringen.

Neben dem Buch sollte sich der Leser zum Studium auch eine Zeichenschablone und geeignete Programmschemen beschaffen. IBM-Handbücher über das System IBM /370 in einer angemessenen Zahl können ihm ebenfalls beim Studium behilflich sein.

St. Paul, Minnesota

Clarence B. Germain

Inhaltsverzeichnis

Vorwort

I. Textteil

1. Die Lochkarte 1

1.1. Hermann Hollerith, Erfinder und Pionier der Lochkartentechnik 1
1.2. Die 80 spaltige Lochkarte 2
1.3. Der Locher IBM 29 3
1.4. Der Kartenlocher und -prüfer IBM 129 9
1.5. Die 96 spaltige Lochkarte 13
1.6. Der Kartenlocher und -prüfer IBM 5496 14
1.7. Andere Lochkartenmaschinen und -geräte 15

2. Einführung in mathematische Bezeichnungen und in die mathematische Schreibweise 18

2.1. Reelle Zahlen 18
2.2. Indizes 19
2.3. Zahlensysteme 20
2.4. Die Gleitpunktdarstellung 24
2.5. Boolesche Arithmetik 25
2.6. Komplementierung 27
2.7. Rundungsfehler 28
2.8. Statistik 30
2.9. Matrizen 32
2.10. Logarithmen 37
2.11. Trigonometrie 38
2.12. Komplexe Zahlen 40

3. Der Aufbau und die Wirkungsweise von Computern 42

3.1. Programme und Programmabläufe 42
3.2. Bytes 44
3.3. Alphanumerische Informationen 46
3.4. Der Code USASCII 48
3.5. Speichereinrichtungen 49
3.6. Felder und Wörter 53
3.7. Seiten und Segmente 54
3.8. Register 55
3.9. Die Steuerung von Computern 56
3.10. Das Datenverarbeitungssystem IBM /370 57

4. Instruktionen 59

4.1. Die Instruktionen der IBM /370 59
4.2. Die Daten in Festpunktdarstellung 61
4.3. Instruktionen zur Verarbeitung von Daten in Festpunktdarstellung 62
4.4. Verzweigungen 69
4.5. Eingabe und Ausgabe (EA) 74
4.6. Instruktionen im SS-Format 79
4.7. Gepackte Daten (Daten in gepackter Form) 81

4.8. Dezimal-Dual-Konvertierung ... 83
4.9. Dezimalarithmetik ... 85
4.10. Programmunterbrechungen ... 90
4.11. Das Programmstatuswort (PSW) ... 92

5. Die Programmierung der Eingabe/Ausgabe ... 98

5.1. Grundlagen der Eingabe/Ausgabe ... 98
5.2. Übertragungsgeschwindigkeiten von Daten ... 102
5.3. Ein Kanalprogramm ... 103
5.4. Die Verbindung zwischen Zentraleinheit und den Kanälen ... 106
5.5. EA-Unterbrechungen ... 110
5.6. Statusbytes und Prüfbytes ... 112
5.7. Einleitendes Programmladen ... 114
5.8. Das Kanalbefehlswort (CCW) ... 115
5.9. Das Kanalstatusbyte ... 119
5.10. Starten Eingabe/Ausgabe (SIO) ... 121
5.11. Starten Eingabe/Ausgabe mit schneller Freigabe (SIOF) ... 124
5.12. Halten Eingabe/Ausgabe ... 125
5.13. Prüfen Kanal (TCH) ... 127
5.14. Prüfen Eingabe/Ausgabe (TIO) ... 127
5.15. Aufheben der Eingabe/Ausgabe (CLRIO) ... 129
5.16. Ein Beispiel für die Programmierung der EA ... 130
5.17. Kanalidentifizierung ... 134

6. Eingabe/Ausgabe - Geräte (EA-Geräte) ... 137

6.1. Das Interface (Schnittstellen) ... 137
6.2. Der Kartenleser IBM 2501 ... 139
6.3. Das Kartengerät IBM 2520 ... 143
6.4. Die Karteneinheit IBM 2540 ... 146
6.5. Die Kartenleser IBM 3504 und IBM 3505 ... 153
6.6. Der Kartenstanzer IBM 3525 ... 159
6.7. Das Lochkartengerät IBM 2596 ... 161
6.8. Das Mehrfunktionskartengerät IBM 5425 ... 164
6.9. Der Konsoldrucker IBM 3215 mit Tastatur ... 168
6.10. Die Konsole IBM 3125 ... 170
6.11. Zeilendrucker ... 174
6.12. Magnetbänder ... 182
6.13. Das Magnetbandgerät IBM 3420 ... 185
6.14. Andere Magentbandgeräte ... 193
6.15. Plattenstapel 194
6.16. Die Datenorganisation bei Speichereinheiten mit direkter Zugriffsmöglichkeit (SEDZ) ... 197
6.17. Die Befehlsschlüssel bei den Speichereinheiten mit direkter Zugriffsmöglichkeit (SEDZ) ... 201
6.18. Das Gerätestatusbyte und die Prüfbytes bei Plattenspeichergeräten ... 209
6.19. Überblick über die Speichereinheiten mit direkter Zugriffsmöglichkeit (SEDZ) ... 212

7. Weitere Instruktionen ... 216

7.1. Zahlen in Gleitpunktdarstellung ... 216
7.2. Instruktionen für Operationen mit Zahlen in der Gleitpunktdarstellung (Gleitpunktinstruktionen) ... 217

7.3. Zahlen in Gleitpunktdarstellung mit erweiterter Genauigkeit 221
7.4. Halbwort-Instruktionen 224
7.5. Instruktionen im RS-Format 224
7.6. Instruktionen im SI-Format 225
7.7. Logische Instruktionen 226
7.8. Weitere Verzweigungsinstruktionen 230
7.9. Stellenverschiebung (Schiften) 233
7.10. Einige weitere Instruktionen verschiedener Art 235
7.11. Privilegierte Instruktionen 236
7.12. Die Hauptspeicherschutzeinrichtung und die für sie geltenden Instruktionen .. 238
7.13. Programmunterbrechungen 241
7.14. Andere Unterbrechungsarten 247
7.15. Byteorientierte Operanden 249
7.16. Instruktionen mit Masken 250
7.17. Übersetzungsinstruktionen 251
7.18. Aufbereitung 252
7.19. Die Instruktion "Ausführen" 257
7.20. Unterbrechbare Instruktionen 258
7.21. Die Instruktionsformate 261

8. Das virtuelle Hauptspeicherkonzept 264

8.1. Das Datenverarbeitungssystem IBM /360 264
8.2. Das Datenverarbeitungssystem IBM /370 265
8.3. Virtuelle Adressen 266
8.4. Dynamische Adreßumsetzung 267
8.5. Der Austausch von Seiten (Seitenwechsel) 271
8.6. Der Umsetzungsablagepuffer 272
8.7. Die Instruktionen zum Ansprechen der dynamischen Adreßumsetzung 272
8.8. Die Identifizierung von Zentraleinheiten 274
8.9. Zeitgeber 275
8.10. Zusätzliche Zeitmeßeinrichtungen 277
8.11. Mehrprozessorsysteme 278
8.12. Die Handhabung der Speicherschutzschlüssel im Programmstatuswort 280
8.13. Instruktionen zur Synchronisierung 280
8.14. Der Monitoraufruf 282
8.15. Die Programmereignis-Registrierung 283
8.16. Die Mikroprogrammierung 284
8.17. Einleitendes Programmladen 285
8.18. Die Bedienung des Steuerpultes 286

9. Grundlagen der Programmierung 290

9.1. Der Aufgabenkreis des Programmierers 290
9.2. Datenflußpläne 291
9.3. Programmablaufpläne 294
9.4. Unterprogramme (Unterroutinen) 298
9.5. Dezimalpunkte 300
9.6. Datenmengen 303
9.7. Die Blockung von Sätzen 305
9.8. Satzarten 307
9.9. Steuerzeichen für Drucker 311
9.10. Bandkennsätze (Bandetiketten) 312
9.11. Kennsätze bei SEDZ 317

10. Betriebssysteme 323

10.1. Programmiersprachen 323
10.2. Betriebssysteme 325
10.3. Steuerprogramme 326
10.4. Dienstprogramme 327
10.5. Objektmoduln 328
10.6. Die logische EA-Einheit des Betriebssystemes DOS/VS 329
10.7. Gerätezuweisungen im Betriebssystem DOS/VS 331
10.8. Die Jobsteuerung (Auftragsteuerung) beim Betriebssystem DOS/VS 335
10.9. Die Steuerstatements für die Standardkennsätze im Betriebssystem DOS/VS 341
10.10. Das Systemprogramm POWER im Betriebssystem DOS/VS 344
10.11. Der Informationsaustausch zwischen Jobstep beim Betriebssystem DOS/VS 345
10.12. Konzepte des Betriebssystems OS 347
10.13. Das JOB-Steuerstatement im Betriebssystem OS 351
10.14. Das EXEC-Steuerstatement im Betriebssystem OS 356
10.15. Das DD-Steuerstatement im Betriebssystem OS 358
10.16. Katalogisierte Prozeduren im Betriebssystem OS 363
10.17. Die konsekutive (starrfortlaufende) Organisationsform für Datenmengen .. 367
10.18. Die wahlfreie (direkte) Organisationsform für Datenmengen 368
10.19. Indexsequentiell (logisch fortlaufend) organisierte Datenmengen 369
10.20. Virtuell sequentiell organisierte Datenmengen 374
10.21. Einige Aspekte zum Aufbau virtuell sequentiell organisierter Datenmengen 376
10.22. VSAM-Dienstleistungen 379

11. Einführung in PL/1 386

11.1. PL/1 386
11.2. Grundlagen der Syntax 388
11.3. Problemdaten 390
11.4. Konstanten und Variablen 394
11.5. Schweibweise (Notation) 397
11.6. Programmsteuerungsdaten 398
11.7. Arithmetische Ausdrücke 399
11.8. Ergibtanweisung 401
11.9. Programmablaufsteuerung 403
11.10. Die listengesteuerte Eingabe/Ausgabe (EA) 405
11.11. Ein Beispielprogramm 409
11.12. Vergleiche 409
11.13. Die datengesteuerte Eingabe/Ausgabe (EA) 413
11.14. Die Anweisung DISPLAY 414
11.15. Die formatgesteuerte Eingabe/Ausgabe (EA) 415
11.16. Druckdateien 422
11.17. Das Eröffnen und Abschließen von Dateien 424

12. Das Schreiben von PL/1-Programmen 426

12.1. ON-Bedingungen 426
12.2. Abkürzungen von Schlüsselwörtern 430
12.3. Anfangswerte 431
12.4. Die Zusammenfassung von Attributen 432
12.5. Eingefügte Funktionen 432

12.6. Indizes ... 434
12.7. Markenvariablen ... 439
12.8. Logische Operationen ... 441
12.9. Statementgruppen ... 447
12.10. DO-Statements zur Einleitung von Schleifen ... 449
12.11. Indizierung in Datenlisten bei Anweisungen der reihenweisen EA ... 452
12.12. Strukturen ... 455
12.13. Abbildungen ... 458
12.14. Die satzweise Eingabe/Ausgabe ... 463
12.15. Behandlung eines Programmbeispieles für die satzweise EA ... 471
12.16. Zeigerdaten ... 474
12.17. Die satzweise EA unter Benutzung von Puffern ... 478
12.18. Die GET- und PUT-Anweisungen mit der Option STRING ... 480
12.19. Die Pseudovariablen ... 481

13. Fortgeschrittene Sprachelemente von PL/1 ... 484

13.1. Das Aufdatieren von Datenmengen bei Benutzung der sequentiellen Zugriffsmethode ... 484
13.2. Das Verarbeiten von Datenmengen ohne Benutzung von Puffern ... 484
13.3. Die Zugriffsmethoden bei Datenmengen mit regionaler Organisationsform ... 485
13.4. Datenmengen mit indexsequentieller Organisationsform ... 488
13.5. Dateien mit den von VSAM unterstützten Dateiorganisationsformen ... 490
13.6. Ein Beispiel für die Direktverarbeitung ... 491
13.7. BEGIN-Blöcke (Einlaufblöcke) ... 498
13.8. Hauptspeicherklassen ... 500
13.9. Funktionen ... 505
13.10. Subroutinen ... 510
13.11. ENTRY-Statements ... 513
13.12. Spezielle Testhilfen ... 514
13.13. Neudefinitionen ... 519
13.14. Die Anordnung der Elemente in Strukturen ... 522
13.15. Datenumwandlungen ... 528
13.16. Ketten mit variabler Länge ... 531
13.17. Gebiete im Hauptspeicher ... 533
13.18. Die gesteuerte Speicherklasse ... 535
13.19. Das Einschließungsstatement zur Vorübersetzerzeit ... 536
13.20. Die Übergabe von Argumenten an PL/1-Hauptprogramme ... 537

14. Einführung in die FORTRAN-Sprache ... 542

14.1. FORTRAN ... 542
14.2. Das Schreiben von FORTRAN-Programmen ... 543
14.3. Konstanten und Variablen ... 546
14.4. Arithmetische Ausdrücke ... 549
14.5. Ergibtanweisungen ... 552
14.6. Steueranweisungen ... 553
14.7. Eingabe und Ausgabe ... 557
14.8. Die FORMAT-Vereinbarung ... 559
14.9. Format von Eingabedaten ... 562
14.10. Ein Beispiel ... 563

15. Zusätzliche FORTRAN-Statements ... 570

15.1. Alphanumerische Formatangaben ... 570
15.2. Vielfache Formatangaben ... 573

15.3. Weitere Formatangaben verschiedener Typen 576
15.4. Indizes 581
15.5. DO-Anweisungen 582
15.6. Weitere Statements 587
15.7. Eingabe/Ausgabe bei Magnetbändern und Magnetplattenspeichern 592
15.8. Die Schleifennotation in Datenlisten von Eingabe/Ausgabe-Anweisungen ... 596
15.9. Namenslisten 598

16. Fortgeschritten FORTRAN-Sprachelemente 602
16.1. Komplexe Variablen 602
16.2. Logische Daten und Statements 604
16.3. Funktionsdefinitionen (Funktionsvereinbarungen) 609
16.4. Funktionsunterprogramme 611
16.5. Unterprogramme vom SUBROUTINE-Type (Subroutine-Unterprogramme) . 616
16.6. Die Vereinbarung COMMON 623
16.7. Anfangswerte 626

17. Die Einführung in COBOL 631
17.1. Die Programmierung kommerzieller Probleme 631
17.2. Der Aufbau eines COBOL-Programmes 633
17.3. Der Zeichenvorrat von COBOL 637
17.4. Wörter und Literale in COBOL 637
17.5. Interpunktionsregeln 639
17.6. Hinweise zur Notation 640
17.7. Der Erkennungsteil 641
17.8. Der Maschinenteil 642
17.9. Die Dateibeschreibung des Datenteils 644
17.10. Die Datenbeschreibung den Datenteils 645
17.11. Der Prozedurteil 647
17.12. Über das Schreiben von COBOL-Programmen 651
17.13. Alphabetische Daten 654
17.14. Alphamerische Daten 655
17.15. Aufbereitete alphamerische Daten 656
17.16. Externe Dezimaldaten 657
17.17. Interne Dezimaldaten 659
17.18. Dualdaten 660
17.19. Aufbereitete numerische Daten 661
17.20. Externe Gleitpunktdaten 664
17.21. Interne Gleitpunktdaten 665
17.22. Das Statement MOVE 665
17.23. Arithmetische Statements 669
17.24. Zwischenergebnisse 671

18. Weitere Sprachelemente von COBOL 674
18.1. Unterteilung von COBOL-Programmen 674
18.2. Kommentare und Fortsetzungen 675
18.3. Spezielle Register 675
18.4. Neudefinitionen (Klausel REDEFINES) 676
18.5. Die Definition von Bereichen (OCCURS-Klausel) 678
18.6. Die Definition von Bereichen mit variablen Dimensionsgrenzen 681
18.7. Eintragungen mit besonderen Datenbeschreibungen 682
18.8. Eintragungen mit der Stufennummer 66 683
18.9. Eintragungen mit der Stufennummer 88 684

18.10. Gekennzeichneter Name 685
18.11. Das Ausrüstungskapitel im Maschinenteil 690
18.12. Der Dateisteuerungsparagraph im EA-Kapitel des Maschinenteils 692
18.13. Der EA-Steuerungsparagraph im EA-Kapitel des Maschinenteils 693
18.14. Eingaben und Anzeigen 695
18.15. Die Formularsteuerung bei Zeilendruckern 697
18.16. Statements für die sequentielle EA 698
18.17. Ein Beispielprogramm 701
18.18. Testhilfestatements 706
18.19. Die Statements GO TO und ALTER 707
18.20. Die Beendigung von Programmen 709
18.21. Das IF-Statement 710
18.22. Vergleichsbedingungen 711
18.23. Klassenbedingungen 713
18.24. Andere Bedingungen 714
18.25. Verbundene Bedingungen 715

19. Sprachelemente von COBOL für den Fortgeschrittenen 718

19.1. Die Option CORRESPONDING 718
19.2. Das Statement EXAMINE 718
19.3. Das Statement TRANSFORM 720
19.4. Das Statement PERFORM 721
19.5. Die SIGN-Klausel im Datenteil 725
19.6. Die Formatsteuerung 725
19.7. Bibliothek mit Quellenstatements 726
19.8. EA-Dateien 728
19.9. Definition und Verarbeitung indexsequentiell organisierter Dateien 729
19.10. Definition und Verarbeitung von Dateien, die nach VSAM organisiert sind 735
19.11. Definition und Verarbeitung von Dateien, die der gesteuerten (direkten) Organisationsform unterliegen 738
19.12. Die Maßnahmekapitel im Prozedurteil 744
19.13. Unterprogramme 749
19.14. Beispiel für ein Unterprogramm 752
19.15. Segmentierung von Programmen 755
19.16. Indexnamen 757
19.17. Indexdatenelemente 760
19.18. Die Anweisung SEARCH 760
19.19. Aufsteigende und fallende Schlüssel 761
19.20. Die Anweisung SEARCH ALL 762
19.21. Die Handhabung von Ketten 764

20. Die Assembliersprache 768

20.1. Das Codieren mit veränderlichen Adressen 768
20.2. Ein Beispiel 770
20.3. Die Assembliersprache für das System IBM /370 774
20.4. Fortsetzung des Beispiels 777
20.5. Glieder (Terme) und Ausdrücke 778
20.6. Die USING-Vereinbarung 781
20.7. Instruktionen 783
20.8. Erweiterte mnemonische Operationsschlüssel 786
20.9. Zeichenkonstanten 789
20.10. Sedezimale und duale Konstanten 791

20.11. Weitere numerische Konstanten 793
20.12. Adreßkonstanten 795
20.13. Literale 797
20.14. Definitionen von Datengebieten 798
20.15. Das CCW-Statement 799
20.16. Steueranweisungen für den Assemblierer 801
20.17. Der Zuordnungszähler 807
20.18. Die Zerlegung von Programmen 810
20.19. Externe Namen 813
20.20. Rettungsgebiete (Sicherungsgebiete) 815
20.21. Der Austausch von Daten zwischen Programmeinheiten 818

21. Makrostatements für das Betriebssystem DOS 823
21.1. Die Programmierung der Eingabe und Ausgabe 823
21.2. Dateidefinitionen 824
21.3. Makrostatements für die sequentielle Verarbeitung 825
21.4. Die Dateidefinition DTFCD 830
21.5. Die Dateidefinition DTFPR 834
21.6. Die Dateidefinition DTFCN 836
21.7. Die Dateidefinition DTFMT 838
21.8. Die Verwendung von Pseudoabschnitten DSECT 844
21.9. Die Dateidefinition DTFSD 848
21.10. Arbeitsdateien 849
21.11. Die Dateidefinition DTFDI 855
21.12. Die Dateidefinition DTFDA 857
21.13. Die Dateidefinition DTFIS 863
21.14. Die Moduln des IOCS 869
21.15. Das physische IOCS 876
21.16. Ein Beispiel 882
21.17. Die Supervisormakroinstruktionen 883
21.18. Speichermakros 887
21.19. Programmverbindungen 888

22. Makrostatements im Betriebssystem OS 892
22.1. Die Programmierung der Eingabe/Ausgabe 892
22.2. Der Datensteuerblock (DCB) 894
22.3. Zugriff zum Datensteuerblock (DCB) 897
22.4. Puffergebiete bei den erweiterten Zugriffsmethoden 898
22.5. Die Zugriffsmethoden QSAM 901
22.6. Die Zugriffsmethoden QUISAM 907
22.7. Einige Beispiele 911
22.8. Die Steuerblöcke DECB, ECB und IOB 916
22.9. Puffer für die Basis-Zugriffsmethoden 919
22.10. Die Zugriffsmethode BSAM 919
22.11. Die Zugriffsmethode BISAM 924
22.12. Die Zugriffsmethode BDAM 926
22.13. Listen- und Ausführungsformen von Makros 929
22.14. Das physische IOCS 930
22.15. Makros für Speicherauszüge 934
22.16. Supervisor-Makroinstruktionen 936
22.17. Makros für den Informationsaustausch mit dem Bedienungspersonal 937
22.18. Makros für Zeitbestimmung 939

22.19. Einfache Programmverbindungen 943
22.20. Dynamische Programmverbindungen 945

23. VSAM 948
23.1. Die Steuerblöcke von VSAM 948
23.2. Das Eröffnen und Schließen von VSAM-Dateien 955
23.3. Das Modifizieren von Steuerblöcken 956
23.4. Auskunft über Steuerblöcke 956
23.5. Das Überprüfen von Werten in Steuerblöcken 957
23.6. Makrostatements für die Verarbeitung 958
23.7. Fehlerbehandlung 961
23.8. Die Erstellung von Dateien 963
23.9. Betrachtung eines Beispieles 965

24. Verschiedenes 970
24.1. Arbeitsgruppen mit Chefprogrammierern 970
24.2. Strukturierte Programmierung 971
24.3. Rekursive Programmierung 973
24.4. Die Definitionen in der Makrosprache 976
24.5. Sortieren 978
24.6. Steuerkarten für das Sortieren 979
24.7. Sortieren unter dem Betriebssystem DOS (Statements der Jobsteuersprache) 984
24.8. Sortieren unter dem Betriebssystem OS (Statements der Jobsteuersprache) 985
24.9. Sortieren in PL/1-Programmen 986
24.10. Durch COBOL-Programme veranlaßtes Sortieren 988

II. Aufgaben

Aufgabengruppe 1 16
Aufgabengruppe 2 23
Aufgabengruppe 3 29
Aufgabengruppe 4 36
Aufgabengruppe 5 41
Aufgabengruppe 6 48
Aufgabengruppe 7 58
Aufgabengruppe 8 68
Aufgabengruppe 9 77
Aufgabengruppe 10 95
Aufgabengruppe 11 115
Aufgabengruppe 12 135
Aufgabengruppe 13 180
Aufgabengruppe 14 214
Aufgabengruppe 15 222
Aufgabengruppe 16 229
Aufgabengruppe 17 239
Aufgabengruppe 18 260
Aufgabengruppe 19 275
Aufgabengruppe 20 288
Aufgabengruppe 21 303
Aufgabengruppe 22 321
Aufgabengruppe 23 346
Aufgabengruppe 24 365
Aufgabengruppe 25 384
Aufgabengruppe 26 402
Aufgabengruppe 27 412
Aufgabengruppe 28 425
Aufgabengruppe 29 445
Aufgabengruppe 30 454
Aufgabengruppe 31 473
Aufgabengruppe 32 483
Aufgabengruppe 33 496
Aufgabengruppe 34 516
Aufgabengruppe 35 538
Aufgabengruppe 36 552
Aufgabengruppe 37 555
Aufgabengruppe 38 565
Aufgabengruppe 39 579
Aufgabengruppe 40 586
Aufgabengruppe 41 600
Aufgabengruppe 42 607
Aufgabengruppe 43 615
Aufgabengruppe 44 628
Aufgabengruppe 45 652
Aufgabengruppe 46 667
Aufgabengruppe 47 673
Aufgabengruppe 48 688
Aufgabengruppe 49 709
Aufgabengruppe 50 717

Aufgabengruppe 51 727
Aufgabengruppe 52 747
Aufgabengruppe 53 766
Aufgabengruppe 54 787
Aufgabengruppe 55 800
Aufgabengruppe 56 820
Aufgabengruppe 57 842
Aufgabengruppe 58 856
Aufgabengruppe 59 889
Aufgabengruppe 60 914
Aufgabengruppe 61 935
Aufgabengruppe 62 946
Aufgabengruppe 63 966
Aufgabengruppe 64 992

III. Anhänge

A. Zeichen und Codes 995
1. Zeichen 995
2. Sedezimalcode 996
3. EBCDI-Code 996
4. ASCII 1001

B. Tabellen aller Instruktionen der IBM /370 1002

C. Übersicht über die Eingabe/Ausgabe 1007
1. Instruktionen, Befehle und Steuerwörter 1007
2. Befehlsschlüssel (ohne SEDZ) 1008
3. Befehlsschlüssel bei SEDZ 1010

D. Referenzdaten 1011
1. Datenformate 1011
2. Werte des Bedienungsschlüssels (BS) 1012
3. Modelle des Systems IBM /370 und ihre Kenngrößen 1013
4. Programmstatuswort (PSW) 1014
5. Aufbereitung 1015
6. Schlüssel für Programmunterbrechungen 1016
7. Steuerungsregister 1017
8. Permanente Hauptspeicherzuordnung 1018

E. Übersicht über PL/1 1020
1. Abkürzung von Schlüsselwörtern 1020
2. PL/1-Statements 1021
3. Attribute in DECLARE-Vereinbarungen 1023
4. Formatelemente 1024
5. Datenumwandlungen 1025
6. Eingefügte Funktionen 1029
7. Pseudovariablen 1031
8. ON-Bedingungen 1033

F. Übersicht über die FORTRAN-Sprache 1036
1. Ausdrücke 1036
2. Datenlisten 1037
3. Indizes 1037
4. Formularsteuerung beim Drucker 1037
5. Formatangaben 1037
6. FORTRAN-Statements 1038
7. Funktionen der FORTRAN-Sprache (offene Unterprogramme, auch "In-Line-Funktionen" genannt) 1043
8. Funktionen der FORTRAN-Sprache (geschlossene Unterprogramme, auch "Out-of-Line-Funktionen" genannt) 1044

G. Reservierte Wörter in COBOL 1045

H. Kenngrößen der SEDZ 1048
1. Spurkapazitäten 1048
2. Übersicht über die Kenngrößen der SEDZ 1050

I. Steuerzeichen 1051

J. Übersicht über die Assemblierersprache (AL) 1052
1. Instruktionsformate 1052
2. Konstanten 1053
3. Erweiterte mnemonische Operationsschlüssel für die Instruktionen BCR und BC 1054
4. Vereinbarungen 1055

K. Übersicht über das Betriebssystem DOS/VS 1056
1. Die normale Prozedur für das einleitende Programmladen (IPL-Prozedur) 1056
2. Jobsteuerbefehle 1057
3. Jobsteuerbefehle und Jobsteuerstatements 1058
4. Jobsteuerstatements 1060

L. Übersicht über das Betriebssystem OS/VS 1064
1. Befehle des Maschinenbedieners (Operateurs) 1064
2. Statements der Jobsteuersprache (JCL) 1068
3. Die Parameter des EXEC-Steuerstatements für katalogisierte Prozeduren 1072

M. Programmiersprachen 1080
1. Maschinenorientierte Sprachen 1080
2. Problemorientierte Sprachen 1080

Literaturverzeichnis 1082

Sachregister 1083

1. Die Lochkarten

1.1. Herman Hollerith, Erfinder und Pionier der Lochkartentechnik

Nachdem Herman Hollerith im Alter von 19 Jahren sein Studium an der Bergbaufakultät der Universität von Columbia abgeschlossen hatte, nahm er 1879 eine Stellung im Amt für die Durchführung von Volkszählungen beim Ministerium des Innern der Vereinigten Staaten von Nordamerika an. Hollerith wurde die Aufgabe übertragen, bei der Volkszählung 1880 an den statistischen Untersuchungen über das Unternehmertum mitzuwirken. Dort begegnete er Dr. John Billings, dem Mann, der mit der Zusammenstellung der Statistik über die Lebenserwartung betraut war. Dieser entwickelte wiederholt vor Hollerith seine Gedanken, daß man für die Abwicklung der rein mechanischen Arbeiten beim Auszählen der verschiedenen Statistiken eine Maschine schaffen müßte. Im Jahre 1882 folgte Hollerith einem Ruf an das Institut für Technologie von Massachusetts, um dort Vorlesungen über Maschinenbau zu halten. Am M.I.T.[1] begann er mit den Experimenten zur Entwicklung einer Zählmaschine. Unbefriedigt von seiner Lehrtätigkeit verließ er das M.I.T., setzte jedoch die konstruktiven Arbeiten an dieser Maschine fort. Während einer Bahnfahrt beobachtete er den Schaffner, wie er mit einer Lochzange die Fahrkarten lochte, die auf gesonderten Feldern die Angaben über die Fahrroute der Reisenden enthielten. Er zog daraus den Schluß, daß man eine Lochkarte entwerfen sollte, um die Untersuchungsergebnisse über die Volkszählung in Amerika festzuhalten. Im Jahre 1884 erwarb er das Patent über seine Zählmaschine. In den darauffolgenden Jahren wurden ihm noch dreißig weitere Patente auf diesem Gebiet zuerkannt.

Das von Hollerith entworfene System ging von der Lochkarte aus und schloß folgende Maschinen ein: Einen Locher, durch den der Bediener in die Lage versetzt wurde, mehr als 100 Lochungen pro Minute durchzuführen, einen Folgestanzer, mit dessen Hilfe es möglich war, zu wiederholende Informationen (Staat, Kreis usw.) nacheinander in eine ganze Folge von Karten zu stanzen, eine Sortiermaschine und eine Zählmaschine. Die Sortiermaschine bestand aus einer ganzen Anzahl von Fächern, von denen jedes seine eigene Zuführung besaß. Die Karten liefen zwischen einer Reihe von unter Federdruck stehenden Stiften und ihren aus Quecksilber bestehenden Gegenkontakten hindurch. Wurde ein Loch gefunden, so berührte der Federdruckstift das Quecksilber und schloß dadurch einen Stromkreis, der wiederum die Zuführung für ein Kartenfach freigab, wodurch die Karte in das ausgewählte Fach abgelegt werden konnte. Die Zählmaschine arbeitete ähnlich wie die Sortiermaschine, nur daß ein entsprechender Zähler um 1 erhöht wurde.

Dieses von Hollerith entwickelte Maschinensystem wurde in den USA bei der Volkszählung von 1890 benutzt. Die Resultate waren wesentlich exakter als die von früheren Volkszählungen, darüberhinaus wurden sie außerdem in nur einem Drittel der Zeit fertiggestellt, die noch für die Auswertung der Volkszählung von 1880 benötigt wurde.

Im Jahre 1896 gründete Hollerith eine Gesellschaft, die sich das Ziel setzte, seine Maschinen auf den Markt zu bringen. 15 Jahre später verkaufte er seine Gesellschaft an eine andere Firma, mit der sie dann zur Computing Tabulating Recording Co. verschmolz. Später änderte dieses Unternehmen seinen Namen in International Business Machines Corporation (IBM)[2].

---

1) M.I.T. ist die Abkürzung von Massachusetts Institute of Technology.

2) IBM - Internationale Büromaschinen Gesellschaft; der Hauptsitz der IBM Deutschland ist Stuttgart, Geschäftsstellen bzw. Niederlassungen befinden sich in allen größeren Städten der Bundesrepublik Deutschland, Fabrikationsstätten u.a. in Berlin, Sindelfingen und Mainz.

1.2. Die 80 spaltige Lochkarte

Abb. 1.1. zeigt eine 80-spaltige Lochkarte der IBM. Sie ist 187,3 mm ($7\frac{3}{8}$ Zoll) lang und 82,6 mm ($3\frac{1}{4}$ Zoll) breit. Sie ist in 80 Spalten unterteilt, die von links nach rechts von 1 bis 80 durchnumeriert werden. Weiterhin ist die Karte in 12 Zeilen eingeteilt, die von oben nach unten wie folgt bezeichnet werden:

12 - 11 (oder X) - 0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9

Rechteckige Löcher können in jede der durch die Spalten- und Zeileneinteilung bedingten (12·80=)960 Stanzpositionen der Karte gelocht werden. Die in einer Spalte getätigten Lochungen repräsentieren gerade ein Zeichen. Die Zeichen werden als numerische Zeichen, alphabetische Zeichen (Buchstaben) und Sonderzeichen klassifiziert. Die 64 Zeichen (einschließlich des Leer- oder Zwischenraumzeichens, auch Blank genannt), die gewöhnlich bei Lochkarten benutzt werden, werden mit ihren Lochkombinationen in Abb. 1.1. vorgestellt. Es existieren jedoch Verschlüsselungen [1] für insgesamt 256 verschiedene Zeichen; diese sind in Anhang A aufgeführt. Auch wenn alle 256 möglichen Zeichen eingesetzt werden, benötigt kein Zeichen mehr als eine Lochung in den Zeilen 1 bis 7.

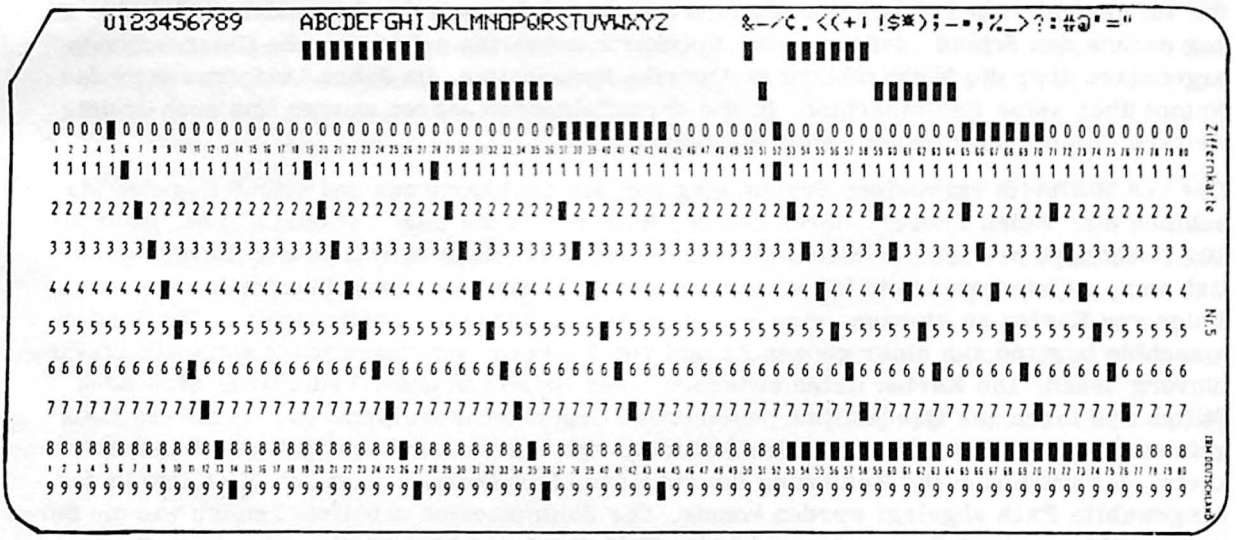


Abb. 1.1. Die 80spaltige Lochkarte

Lochungen in den Zeilen 12, 11 und 0 werden auch als Zonenlochungen bezeichnet, Lochungen in den anderen Zeilen gelten als Ziffernlochungen.

Aus der Abb. 1.1. ist ersichtlich, daß alle Ziffern (Zahlzeichen) nur durch eine einzige Lochung dargestellt werden, die Buchstaben erfordern zwei Lochungen. Für die Sonderzeichen sind entweder keine Lochungen (Leerzeichen), eine Lochung (Zeichen & bzw. -), zwei Lochungen (/ bzw. = usw.) oder drei Lochungen (§ bzw. + usw.) notwendig. Wenn alle 256 möglichen Verschlüsselungen gestanzt werden, so gibt es darunter einige Zeichen, die nicht weniger als sechs Lochungen in der ihnen zugewiesenen Spalte aufweisen.

1) Dieser Code wird nach H. Hollerith auch Hollerithcode genannt.

Als Feld wird bei einer Lochkarte eine Gruppe aufeinanderfolgender Kartenspalten bezeichnet, die zusammen einen bestimmten Informationsinhalt darstellen. Beispiele hierfür zu nennen, fällt nicht schwer: Sie schließen Datum, Betrag, Adresse usw. ein. Wenn numerische Werte (Zahlen) in eine Lochkarte gestanzt werden, so wird das Vorzeichen, um Platz zu sparen, gewöhnlich als Zonenlochung über der Einerstelle der Zahl abgelocht. Eine Lochung in der Zeile 12 repräsentiert dabei einen positiven, eine in der Zeile 11 einen negativen Wert. Das Fehlen einer 12er bzw. 11er Lochung zeigt einen vorzeichenlosen Wert an; dieser wird in üblicher Weise als positiv betrachtet.

1.3. Der Locher IBM 29

Die Abb. 1.2. zeigt einen Locher des Typs IBM 29. Die ins Kartenmagazin eingelegten Karten passieren eine Loch- und eine Lesestation, bevor sie der Kartenablage zugeführt werden. Die in Abb. 1.3. gezeigte Tastatur ist schreibmaschinenähnlich und weist oberhalb der eigentlichen Tastatur verschiedene Kippschalter auf. Die durch Anschlag der einzelnen Tasten erzielten Lochkombinationen werden durch die entsprechenden Zeichen auf den Tasten dargestellt. Sind auf einer Taste zwei Zeichen markiert, so wird die Lochkombination des oberen Zeichens gelocht, wenn die Tastatur auf numerische Arbeitsweise eingestellt ist, bei Einstellung auf die alphabetische Arbeitsweise (kurz: Alpha-Arbeitsweise) wird die Lochkombination gelocht, die dem unteren Zeichen entspricht. Es sollte ausdrücklich betont werden, daß die Ziffern 0 bis 9 auf der Lochertastatur an anderen Stellen angeordnet sind als bei den meisten Schreibmaschinen. Ihre Anordnung ist allein dadurch bestimmt, daß damit die Möglichkeit geboten werden soll, rein numerische Informationen auch einhändig lochen zu können.

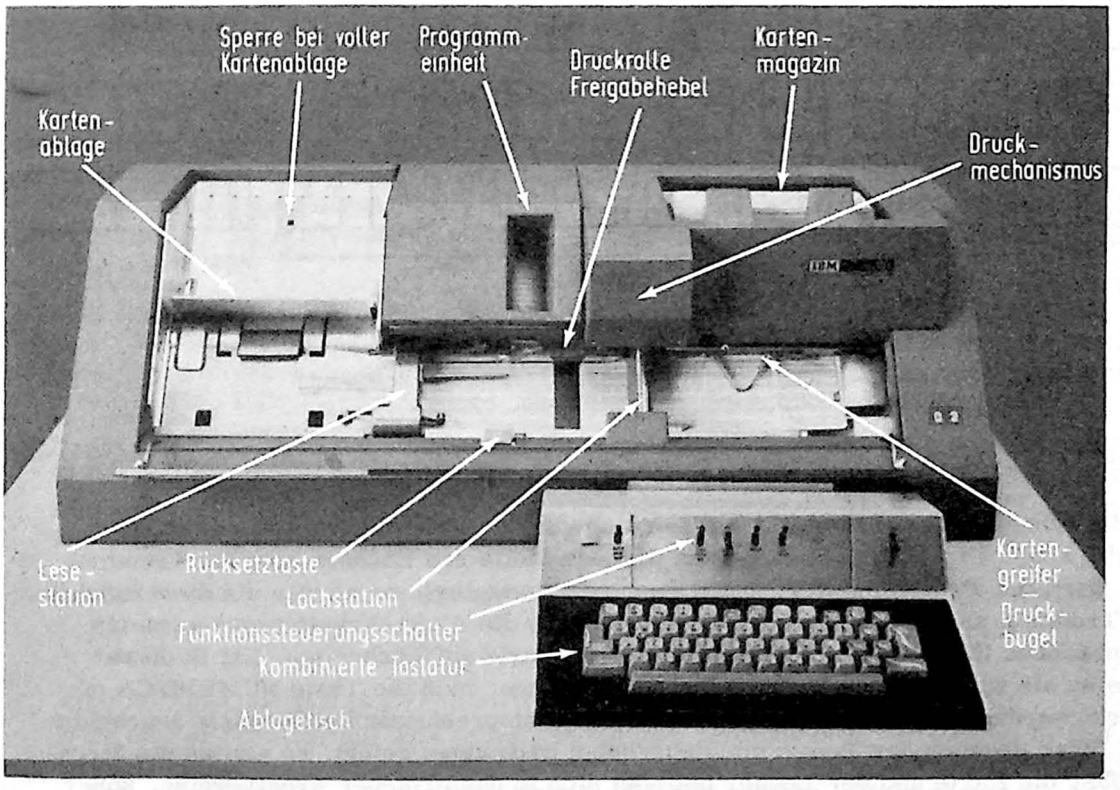


Abb. 1.2. Locher IBM 29

Abb. 1.3. Tastatur oder IBM 29 (Deutsche Belegung der Tastatur siehe Abb. 1.4.)

Punkt, Komma und Bindestrich (Semikolon) sind auf der Tastatur zweimal aufgeführt; somit können diese Zeichen sowohl bei numerischer als auch bei alphabetischer Arbeitsweise gelocht werden. Die Lochkombinationen sind völlig gleich, welche Taste auch angeschlagen wird. Alle 63 Zeichen, die in Abb. 1.1. gezeigt wurden, können mit einem Locher dieses Typs gelocht werden.

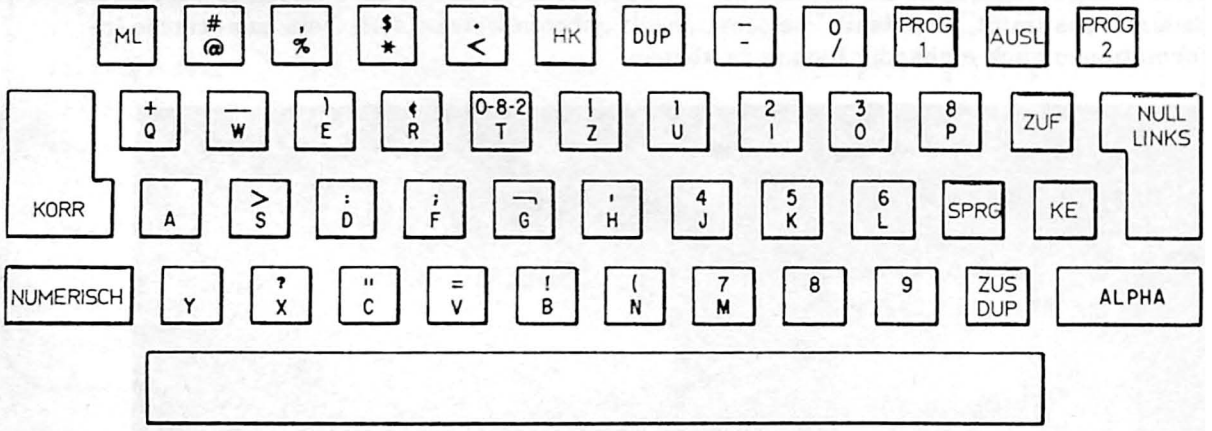


Abb. 1.4. Schematische Darstellung der Tastatur der IBM 29 (deutsche Belegung)

Die Arbeitsweise des Lochers wird hauptsächlich von der Programmtrommel gesteuert (siehe Abb. 1.5.). Die Abb. 1.6. zeigt, an welcher Stelle des Lochers sich die Trommel befindet. Wird der Programmsteuerhebel nach oben umgelegt, so werden die Sternrädchen über die Trommel angehoben, und der Locher wird in die Alpha-Arbeitsweise versetzt. (Die Sternrädchen fühlen sonst die Programmsteuerkarte auf Löcher ab). Um in dieser Arbeitsweise ein numerisches Zeichen lochen zu können, muß die Taste NUMERISCH niedergedrückt werden, während zur gleichen Zeit die entsprechende Zeichentaste angeschlagen wird. Wird dagegen der Programmsteuerhebel nach unten gelegt, so gleiten die Sternrädchen über die Karte und der Locher befindet sich in numerischer Arbeitsweise, außer wenn Lochungen in der Steuerkarte etwas anderes vorschreiben.

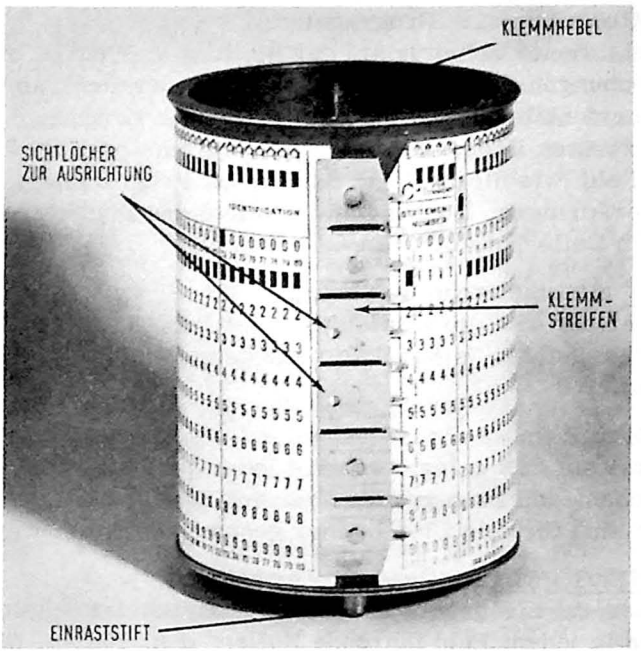


Abb. 1.5. Programmtrommel bei der IBM 29

Zur Steuerung der mechanischen Locherfunktionen muß die Programmsteuerkarte[1)] wie folgt abgelocht werden:

a) Durch Lochungen in Zeile 12 sind alle Spalten eines Feldes mit Ausnahme der ersten zu kennzeichnen.

b) Eine Lochung in Zeile 1 einer Spalte bewirkt, daß der Locher für diese Spalte auf die Alpha-Arbeitsweise eingestellt wird. Wird eine 1 nicht gelocht, so können Alphazeichen nur dann gelocht werden, wenn gleichzeitig mit dem Anschlag der Zeichentaste auch die Taste ALPHA niedergedrückt wird.

c) Eine Lochung in Zeile 11 einer Spalte hat zur Folge, daß der Locher von dieser Spalte ab beginnt, Spalten zu überspringen; das Überspringen setzt sich bis zum Ende des Feldes fort (das Feldende wird durch eine abschließende Lochung in Zeile 12 entdeckt). Das Überspringen bewirkt, daß ein Feld übergangen wird, ohne daß irgendeine Lochung erfolgen kann.

d) Durch eine Lochung in Zeile 0 einer Spalte beginnt der Locher zu duplizieren; die Löcher in der Karte, die die Lesestation passiert, bewirken, daß die Karte, die sich unter der Lochstation befindet, ebenfalls an den gleichen Stellen gelocht wird. Am Ende des Feldes hört das Duplizieren auf.

Durch die oberhalb der Tastatur angebrachten Schalter kann man bestimmte weitere Locherfunktionen regeln:

a) AUTOM SPRG DUP - EIN (Automatisches Springen und Duplizieren)
Wird dieser Schalter auf Aus gelegt, so bleiben Lochungen in den Zeilen 11 und 0 der Programmsteuerkarte unbeachtet.

1) Auch kurz als Programmkarte bezeichnet

b) PROG1 - PROG2 (Programm 1 - Programm 2)
Normalerweise steht dieser Schalter auf der Stellung 1. Wird er auf die Stellung 2 umgelegt, so werden Lochungen in den Zeilen 12 bis 3 auf der Programmkarte ignoriert, zur Steuerung der Locherfunktionen werden stattdessen die Lochungen in den Zeilen 4 bis 9 herangezogen. Im zweiten Programm (dem Alternativprogramm) kennzeichnen Lochungen in Zeile 4 ein Feld (wie die Zeile 12 des ersten Programmes), Lochungen in Zeile 5 bewirken das Überspringen, Lochungen in Zeile 6 das Duplizieren und schließlich bedeuten Lochungen in Zeile 7 die ALPHA-Arbeitsweise.

c) AUTOM ZUF - EIN (Automatische Zuführung)
Steht dieser Schalter auf EIN, so werden nach den beiden ersten Karten alle anderen Karten automatisch zugeführt, sobald die vorangegangene Karte die Lochstation vollständig passiert hat.

d) SCHREIB - EIN (Schreiben)
Wird dieser Schalter auf EIN gelegt, so wird jedes gelochte Zeichen auf dem oberen Kartenrand angeschrieben, und zwar gerade über der Spalte, in die es gelocht wurde. Die Schreibeinrichtung muß für alle 29er Locher gesondert bestellt werden.

e) NULL SCHREIB LINKS - EIN (Nullenschreibung links)
Wird dieser Schalter auf EIN gelegt und steht zusätzlich der Schalter SCHREIB ebenfalls auf EIN, so werden in jedem Feld führende Nullen, d.h. Nullen, die vor der ersten Null verschiedenen Ziffer eines Feldes auftreten, ebenfalls angeschrieben. Meist wird dieser Schalter auf Aus gelegt, wodurch das Schreiben führender Nullen unterdrückt wird.

f) KARTENAUSLAUF (Kartenauslauf)
Wird dieser Schalter angehoben, so werden alle im Kartenbett befindlichen Karten in die Kartenablage transportiert, die Zuführung weiterer Karten unterbleibt.

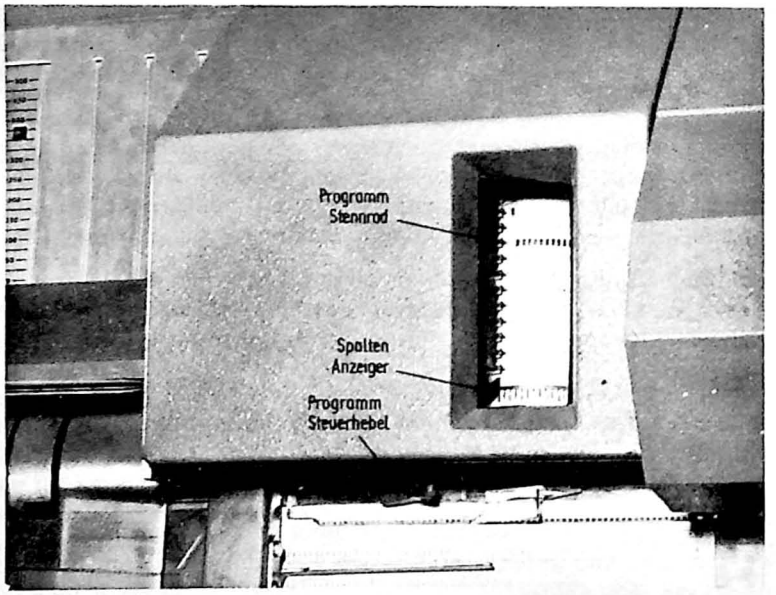


Abb. 1.6. Sternräder bei der IBM 29

Die Taste BACKSPACE (siehe Rücksetztaste in Abb. 1.2.) bewirkt, wenn sie niedergedrückt wird, daß die Karten an der Lesestation und an der Lochstation zurückgesetzt werden; auch die Programmtrommel mit der Programmkarte läuft in gleicher Weise rückwärts. Die Spalte, in die das nächste Zeichen gelocht werden kann, kann an einem Spaltenanzeiger abgelesen werden, der unterhalb der Programmtrommel angebracht ist (siehe Abb. 1.6.).

Um den Aufbau einer Programmkarte zu demonstrieren, wollen wir eine Programmkarte entwerfen, die das Ablochen eines Datenkartenpaketes[1] steuern soll, dessen einzelne Karten folgende Einteilung besitzen sollen:

Spalten 1 bis 5:	Numerisches Feld
Spalte 6:	Soll stets übersprungen werden
Spalten 7 bis 72:	Alpha-Feld (ALPHA-Modus)
Spalten 73 bis 76:	Alpha-Feld, das außerdem von Karte zu Karte dupliziert werden soll
Spalten 77 bis 80:	Sollen stets übersprungen werden

Karten, die gemäß dieser Einteilung abgelocht werden können, können wir durchaus verwenden, wenn wir uns später über die Ablochung von FORTRAN-Programmen unterhalten. Die Programmkarte zur Lösung unserer Aufgabenstellung wird jedenfalls wie folgt abzulochen sein:

Spalte 1:	bleibt leer
Spalten 2 bis 5:	Lochungen in Zeile 12
Spalte 6:	Lochung in Zeile 11
Spalte 7:	Lochung in Zeile 1
Spalten 8 bis 72:	Lochungen in den Zeilen 12 und 1
Spalte 73:	Lochungen in den Zeilen 0 und 1
Spalten 74 bis 76:	Lochungen in den Zeilen 12 und 1
Spalte 77:	Lochung in Zeile 11
Spalten 78 bis 80:	Lochungen in Zeile 12

Das Fehlen von 12er Lochungen in den Spalten 1, 6, 7, 73 und 77 kennzeichnet diese Spalten als die Anfangsspalten von Feldern.

Um unsere Steuerkarte auf die Programmtrommel zu spannen, legen wir zunächst den Klemmhebel bis zum Anschlag um, d.h. so weit er überhaupt geht und ziehen dann die alte Karte von der Trommel ab. Anschließend schieben wir die neue Programmkarte mit der rechten Seite (Seite, auf der Spalte 80 liegt) unter die angelüftete Kante des Klemmstreifens (Seite mit den Sichtlöchern). Danach ist der Hebel um die Hälfte zurückzudrehen, wodurch der Rand der Karte mit der 80er Spalte festgeklemmt wird, während die andere Seite des Klemmstreifens angehoben wird. Wir schieben jetzt die Kartenseite mit der Spalte 1 unter den Metallstreifen, wobei wir uns bemühen müssen, die Karte stets straff zu halten. Zum Schluß bewegen wir den Hebel in seine ursprüngliche Stellung zurück; damit ist die Karte auf der Trommel unverrückbar eingespannt. Die Trommel wird auf die Spindel der Programmeinheit gesteckt und die Sternrädchen können abgesenkt werden. Diese Rädchen sollten jedoch niemals heruntergelassen werden, wenn keine Karte in die Trommel eingespannt ist, weil dadurch der Locher leicht beschädigt werden kann.

1) Unter einem Kartenpaket versteht man eine Menge von Lochkarten; Kartenpakete werden in der Praxis mitunter auch als Kartensätze bezeichnet.

Um die erste zu lochende Karte zuzuführen, müssen wir die Taste ZUF zweimal niederdrücken. Wir legen den Schalter ZUF auf EIN und den Schalter AUTOM SPRG DUP auf aus. Danach lochen wir die Spalten 1 bis 5, überspringen die Spalte 6, lochen die Spalten 7 bis 72 und schlagen die Taste SPRG an, wenn wir das letzte Zeichen dieses Feldes gelocht haben; der Rest dieses Feldes wird dadurch übersprungen. Anschließend lochen wir die Information für die Spalten 73 bis 76 ab. Nun stellen wir den Schalter AUTOM SPRG DUP auf EIN. Dadurch wird ein Sprung bis zum Kartenende hervorgerufen und außerdem wird die zweite Karte zugeführt.

Bei der zweiten und den nachfolgenden Karten lochen wir zunächst das numerische Feld mit den Spalten 1 bis 5 ab. Die Programmkarte ruft anschließend ein automatisches Überspringen der Spalte 6 hervor. Danach lochen wir wieder Informationen in die Spalten 7 bis 72. Füllen diese Informationen nicht das gesamte Feld aus, so können wir die Taste SPRG drücken, und wir erreichen dadurch, daß die restlichen Spalten bis einschließlich Spalte 72 übersprungen werden. Die Spalten 73 bis 76 werden hinterher automatisch dupliziert.

Nach dem Ablochen der letzten Karten können wir den Hebel KARTENAUSLAUF anheben, wodurch der Rest der Karten ausläuft, so daß das Kartenbett frei wird.

Alle Tasten, mit denen die verschiedenen Zeichen abgelocht werden, sind im grauen Farbton gehalten, die Funktionstasten dagegen sind blau. Einige der Funktionstasten haben wir bereits besprochen, die anderen wirken wie folgt:

a) ML (Mehrfachlochung)
Wird diese Taste niedergedrückt, so ist es möglich, beliebig viele Lochungen in einer Spalte unterzubringen, da der Kartentransport während des Niederdrückens unterbleibt. Man braucht nur die entsprechenden numerischen Zeichentasten anzuschlagen. Diese Methode dient dazu, Lochkombinationen zu erzeugen, die durch das Anschlagen der normalen Zeichentasten nicht hervorgerufen werden können.

b) PROG 1 bzw. PROG 2 (Programm 1 bzw. Programm 2)
Wird mit dem Ablochen einer Karte begonnen, so arbeitet dasjenige Programm der Programmkarte, das durch die Stellung des Schalters PROG1 - PROG2 angezeigt wird. Wird nun eine der beiden Tasten PROG1 oder PROG2 gedrückt, so beginnt das Programm die Steuerung der weiteren Ablochung zu übernehmen, dessen Funktionstaste angeschlagen wurde. Diese Programmauswahl bleibt bis zum Kartenende erhalten, nur das Anschlagen der anderen Taste hat eine erneute Programmänderung zur Folge.

c) AUSL (Auslauf)
Das Drücken der Auslauftaste hat zur Folge, daß die unter der Lochstation liegende Karte diese sofort verläßt.

d) KE (Karteneinzelgang)
Das Anschlagen dieser Funktionstaste hat zur Folge, daß die nächste Karte unter die Lochstation gezogen wird; dadurch ist sie ablochbereit. Steht der Schalter AUTOM ZUF auf EIN, so erfolgen Auslauf der alten Karte und Einzug der neuen Karte automatisch beim Niederdrücken von AUSL. Das Anschlagen von KE erübrigt sich dann.

e) DUP (Duplizieren)
Das Niederdrücken dieser Dupliziertaste bewirkt, daß die unter der Lochstation befindliche Kartenspalte selbsttätig mit den Lochungen versehen wird, die die entsprechende Spalte der vorhergehenden Karte aufweist. Wird eine Programmkarte verwendet, so wird das Duplizieren bis zum Ende des Feldes fortgesetzt, ohne Programmkarte hingegen wird solange dupliziert, solange die Taste niedergedrückt wird. Liegt keine Karte unter

der Lesestation, so bleiben die Kartenspalten der unter der Lochstation liegenden Karte leer; auf diese Weise erreicht man schnell und bequem das Überspringen einer bestimmten Anzahl von Spalten von Karten, die einzeln zu lochen sind.

f) ZUS DUP (Zusatz Duplizieren)
Diese Taste ist nur mit einer besonderen Zusatzeinrichtung funktionsfähig. Mit ihrer Hilfe können Informationen von einer zweiten Programmtrommel in die Karte unter der Lochstation dupliziert werden.

g) NULL LINKS und KORR (Nullen links und Korrektur)
Diese Taste dient dazu, das Ablochen von Zahlen mit unterschiedlicher Stellenzahl zu erleichtern, die rechtsbündig in ein numerisches Feld gestellt werden sollen. Die betreffende Zahl wird wie gewöhnlich eingetastet, jedoch ohne die führenden Nullen. Die Taste NULL LINKS wird gedrückt, wodurch die Nullen automatisch hinzugefügt werden und die Zahl gelocht wird. Ist die Zahl negativ, so bewirkt das Drücken der Taste —, daß die Nullen eingesteuert werden und außerdem eine 11er Lochung über der Einerstelle erfolgt. Wird ein Fehler gefunden, so löscht das Drücken der Taste KORR (Korrektur) die Ziffern vollständig, die bereits eingetastet, doch noch nicht gelocht sind. Der Umfang eines solchen „NULL-LINKS-Feldes" wird durch Lochungen in den Zeilen 1, 2 und 3 der Programmkarte angezeigt (bzw. in den Zeilen 7, 8 und 9 beim zweiten Programm). Die Verschlüsselungen für die Programmkarte sind wie folgt vorzugeben:

Feldbreite (Anzahl Spalten)	Programm 1	Programm 2
8	2	8
7	3	9
6	2 und 3	8 und 9
5	1 und 2	7 und 8
4	1 und 3	7 und 9
3	1 und 2 und 3	7 und 8 und 9

Alle anderen Spalten der Programmkarte mit Ausnahme der soeben aufgeführten ersten Spalte sind für ein numerisches Feld mit führenden Nullen wie üblich entweder mit einer Lochung in der 12er Zeile (Programm 1) oder einer Lochung in der 4er Zeile zu kennzeichnen. Felder mit führenden Nullen, die weniger als 3 bzw. mehr als 8 Stellen umfassen sollen, können nicht auf diese Weise abgelocht werden.

Die Einrichtung zur Einsteuerung führender Nullen in numerische Felder ist keine Standardeinrichtung, sie muß bei der IBM gesondert geordert werden.

Bei Lochern, die mit der Schreibeinrichtung ausgerüstet sind, sollten andere als die 63 Standardzeichen nicht dupliziert werden, wenn der Schalter SCHREIB auf EIN gestellt ist. Es könnte sich sonst leicht als Folge eine Beschädigung des Schreibmechanismus einstellen.

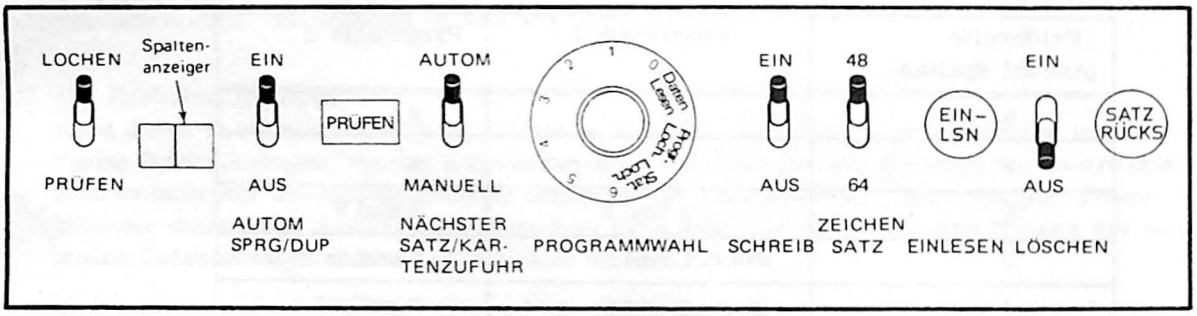
1.4. Der Kartenlocher und -prüfer IBM 129

Die IBM 129 ähnelt der IBM 29 in Aufbau und Bedienungsweise. Lochstation und Lesestation sind jedoch bei ihr zu einer Loch-/Lesestation zusammengefaßt. Sie sollte immer dann eingesetzt werden, wenn entweder eine höhere Arbeitsgeschwindigkeit oder mehr Fähigkeiten verlangt werden als die IBM 29 anbietet. Die IBM 129 besitzt

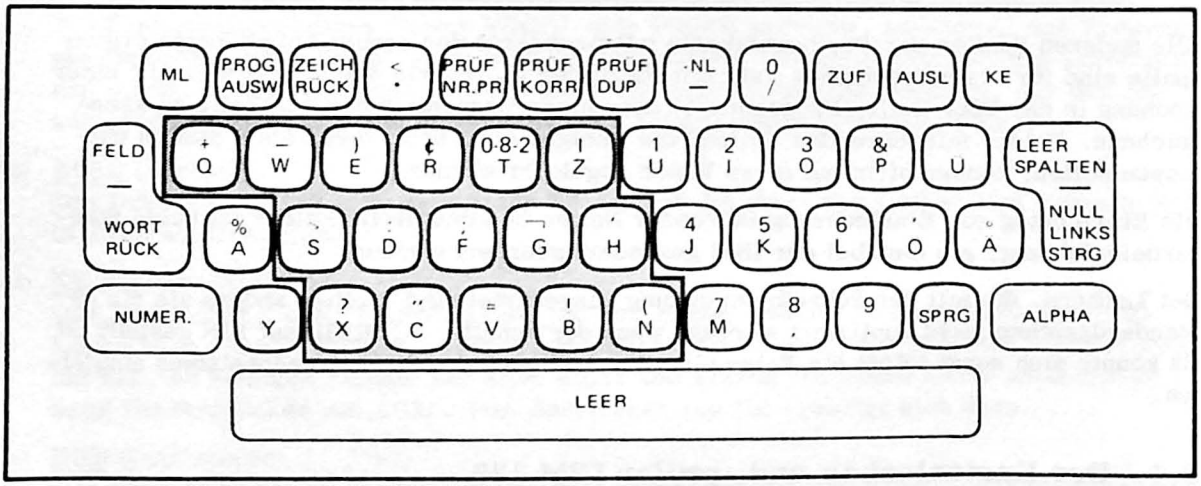
eine elektronische Speichereinheit für Programme und Eingabezeichen. Die vom Bediener bzw. der Bedienerin eingetasteten Daten werden zunächst in diesen Speicher hineingestellt. Erst, wenn die gesamten für eine Karte bestimmten Zeichen eingetastet sind, beginnt die IBM 129 automatisch zu stanzen. Inzwischen kann bereits der Bediener bzw. die Bedienerin die zur darauffolgenden Karte gehörenden Zeichen eintasten. Sprünge und Feldduplizierungen laufen mit elektronischen und nicht wie bei der IBM 29 mit mechanischen Geschwindigkeiten ab.

Sollte ein Bediener bzw. eine Bedienerin einen Fehler entdecken, so kann er bzw. sie den Spaltenanzeiger mittels Tastendruck auf das fehlerhafte Feld bzw. die fehlerhafte Spalte zurücksetzen und die falschen Daten erneut eintasten. Da die meisten Eintastfehler sofort, d.h. noch während des Eintastens, festgestellt werden, bedeutet diese Vorgehensweise eine erhebliche Zeiteinsparung, da keine Zeit mehr für das Duplizieren einer fehlerhaften Karte benötigt wird, womit man sich immerhin bei der Benutzung einer IBM 29 herumschlagen muß.

Jedes numerische Feld kann als Feld mit führenden Nullen angesehen werden. Es gibt für solche Felder keine Längenbeschränkung wie bei der IBM 29. Zusätzlich kann die Taste LEERSPALTEN — NULL LINKS STRG [1] auch für alphabetische Felder benutzt werden, um die für sie vorgesehenen Daten rechtsbündig auszurichten und führende Leerzeichen einzusteuern.



a) Schalterplatte über der Tastatur



b) Tastatur (deutsche Belegung)

Abb. 1.7. Steuerpult der IBM 129

1) Diese Taste steuert demnach sowohl das Voransetzen von Leerzeichen als auch das von Nullen. STRG ist die Abkürzung von Steuerung.

Sechs Programmstufen sind in die IBM 129 eingebaut. Bei den Programmkarten werden die Zeilen 12, 11, 0 und 1 genau so behandelt wie bei den Programmkarten für die IBM 29 (siehe Abschnitt 1.3.). Die Abb. 1.7. zeigt die Funktionssteuereinrichtungen der IBM 129. Um ein Programm zu laden, sind der Reihe nach folgende Handgriffe vorzunehmen:

1) Falls das Kartenbett nicht leer ist, ist der Kipphebel LÖSCHEN zu betätigen.
2) Der Schalter AUTOM SPRG/DUP ist auf AUS zu stellen.
3) Die gewünschte Programmstufe (1 bis 6) ist mittels des Drehschalters PROGRAMMWAHL einzustellen.
4) Die Programmkarte wird manuell in die Loch-/Lesestation eingeführt, wobei es gleichgültig ist, ob die Betriebsart LOCHEN oder PRÜFEN eingestellt ist. An Stelle der manuellen Einführung kann die Karte auch mittels Tastendruck vom Magazin bis vor die Loch-/Lesestation gebracht werden (Betriebsart LOCHEN).
5) Es ist die Einlesetaste EINLSN zu drücken.

Für jede zu speichernde Programmkarte sind die Schritte 3) bis 5) zu wiederholen. Die auf diese Weise eingelesenen Programme bleiben solange im Programmspeicher gespeichert, bis ein neues Programm für diese Stufe geladen wird oder die Maschine abgeschaltet wird. Es ist ferner eine spezielle Programmstufe 0 vorgesehen. Diese entspricht einer Programmkarte, die auf jeder Spalte eine Lochung in der Zeile 1 enthält.

Auch eine Datenkarte kann in den elektronischen Speicher geladen werden. Hierzu muß der Schalter für den Programmodus auf die Stellung DAT LES anstelle auf eine der Programmstufen 1 bis 6 eingestellt werden. Die Datenkarte ist in die Zuführung vor der Loch-/Lesestation einzulegen und danach die Einlesetaste EINLSN zu drücken. Auf diese Weise kann die Karte wie gewünscht dupliziert werden.

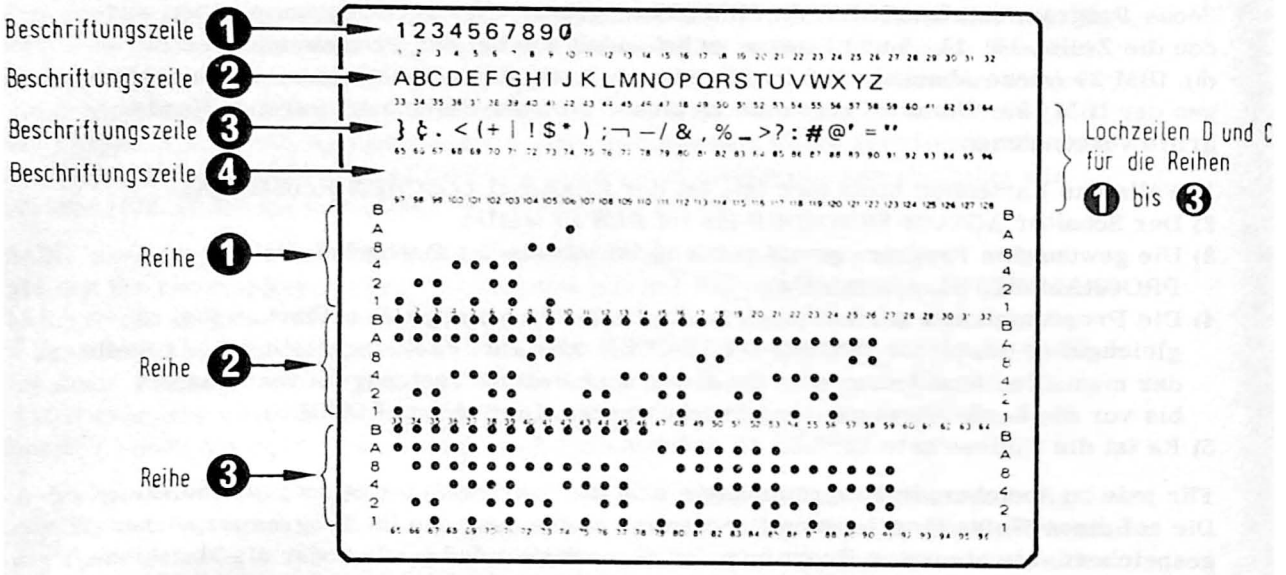
Die Taste SATZ RÜCKS [1] bewirkt das Löschen aller für die laufende Karte eingetasteten Zeichen, stellt den Spaltenanzeiger auf die erste Stelle des Eingabespeichers und ignoriert alle Markierungen [2] in den Zeilen 11 und 0, die in der betreffenden Programmstufe für die Spalte 1 vorliegen.

Durch den Programmdrehschalter wird das zu benutzende Programm vor Beginn des Eintastens der Daten für eine Karte eingestellt. Während des Eintastvorganges hat der Bediener bzw. die Bedienerin die Möglichkeit, beliebig oft Programme zu wechseln, d.h. das eingestellte mit einem anderen zu kombinieren. Zu diesem Zweck ist zunächst die Taste PROG AUSW anzuschlagen und danach eine der numerischen Tasten 0 bis 6. Die Taste PROGR AUSW ersetzt somit die Tasten mit den Bezeichnungen PROG1 bzw. PROG2 bei der IBM 29.

Die IBM 129 besitzt einige weitere Tasten, die gegenüber der IBM 29 eine unterschiedliche Funktionsweise besitzen. Die Taste ZEICH RÜCK[3] z.B. setzt den Speicher um eine Stelle zurück, die Taste FELD —— WORT RÜCK [4] um ein Feld. Wird die Taste — bei numerischer Arbeitsweise niedergedrückt, so hat sie bei Feldern mit führenden Nullen die Erzeugung einer 11er Lochung auf der Einerstelle zur Folge. Diese Taste in alphabetischer Arbeitsweise angeschlagen, führt sie wie bei der IBM 29 zum Lochen eines Minuszeichens (11er Lochung).

---

1) Abkürzung von „Satz rücksetzen".
2) Gemeint sind hier natürlich die Lochungen in den Zeilen 11 und 0 der in den Speicher geladenen Programmkarte der benutzten Programmebene, d.h. die Lochungen, die für das Springen und Duplizieren verantwortlich sind (siehe Abschnitt 1.3.).
3) Abkürzung von „Zeichenrücksetzung",
4) Abkürzung von „Feld/Wort Zurücksetzung".



a) Inhalt der Reihe 1

Numerische Zeichen (Ziffern)	1	2	3	4	5	6	7	8	9	0
Zonenlochungen B										
Zonenlochungen A										A
Ziffernlochungen 8								8	8	
Ziffernlochungen 4				4	4	4	4			
Ziffernlochungen 2		2	2			2	2			
Ziffernlochungen 1	1		1		1		1		1	

b) Inhalt der Reihe 2

Buchstaben	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Zonenlochungen B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B								
Zonenlochungen A	A	A	A	A	A	A	A	A	A										A	A	A	A	A	A	A	A
Ziffernlochungen 8								8	8								8	8							8	8
Ziffernlochungen 4				4	4	4	4					4	4	4	4						4	4	4	4		
Ziffernlochungen 2		2	2			2	2			2	2			2	2				2	2				2		
Ziffernlochungen 1	1		1		1		1		1	1	1		1		1		1			1		1		1		1

c) Inhalt der Reihe 3

Sonderzeichen	}	¢	.	<	(	+	!	$	*	)	;	¬	-	/	&	,	%	_	>	?	:	#	@	'	=	"
Zonenlochungen B	B	B	B	B	B	B	B	B	B	B	B	B	B	B													
Zonenlochungen A	A	A	A	A	A	A	A	A							A	A	A	A	A	A	A						
Ziffernlochungen 8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	
Ziffernlochungen 4				4	4	4	4				4	4	4				4	4	4	4			4	4	4	4	
Ziffernlochungen 2	2	2	2			2	2	2	2		2	2	2		2	2			2	2	2	2		2	2	2	
Ziffernlochungen 1		1		1		1	1	1	1		1	1	1	1	1	1		1		1	1	1		1	1	1	

Abb. 1.8. Die 96spaltige Lochkarte mit der Verschlüsselung der Zeichen

1.5. Die 96 spaltige Lochkarte

Im Jahre 1969 führte die IBM eine neue Lochkarte mit 96 Spalten ein. Diese Karte ist nur 82,6 mm (3,25 Zoll) lang und 66,8 mm (2,63 Zoll) breit, d.h. sie ist nur so lang wie die 80 spaltige Lochkarte (siehe Abschnitt 1.2.) breit ist. Auf ihr stehen insgesamt 96 Spalten zur Aufnahme von Daten zur Verfügung die jeweils in drei übereinanderliegenden Reihen auf der Karte angeordnet sind. Die Abb. 1.8. zeigt eine solche 96 spaltige Lochkarte sowie die Verschlüsselung der Zeichen.

Die 96 spaltige Lochkarte weist keine rechteckigen Löcher wie die 80 spaltige auf. Wie die Abb. 1.8. zeigt, sind es runde Löcher mit dem außerordentlich geringen Durchmesser von 1,2 mm (0,046 Zoll). An Stelle des Hollerithcodes ist ein neuer Code getreten.

Um 256 verschiedene Zeichen zu verschlüsseln, werden genau acht Lochpositionen[1] benötigt. Ein aus 64 Zeichen bestehender Zeichenvorrat, d.h. eine Untermenge aller Zeichen, reicht für viele Zwecke aus; diese Untermenge findet man deshalb in 1.8. auch abgebildet.

Lfd. Zeilennummer	Reihe	Lochzeile	Bemerkungen
1	1	D	
2	2	D	3. Druckzeile
3	3	D	
4	1	C	
5	2	C	4. Druckzeile
6	3	C	
7	1	B	
8	1	A	
9	1	8	
10	1	4	
11	1	2	
12	1	1	
13	2	B	
14	2	A	
15	2	8	
16	2	4	
17	2	2	
18	2	1	
19	3	B	
20	3	A	
21	3	8	
22	3	4	
23	3	2	
24	3	1	

Abb. 1.9. Anordnung der Lochzeichen bei den 96spaltigen Lochkarten

1) Acht Positionen ermöglichen gerade 2^8=256 verschiedene Kombinationen, wenn man alle möglichen Kombinationen zuläßt, d.h. keine Beschränkung der Zahl der Löcher pro Spalte vorgibt.

Der obere Teil der Karte gestattet die Aufnahme von vier Beschriftungszeilen (Druckzeilen) mit jeweils 32 Druckpositionen pro Zeile. Die Lochpositionen überlappen die dritte und vierte Druckzeile dergestalt, daß bei ausschließlicher Verwendung der erwähnten Untermenge von 64 Zeichen kein einziges Zeichen eine Lochung im Bereich einer Druckzeile aufweist. Der untere Teil der Karte wird allein für Lochzwecke benutzt.

Jede Spalte weist wie geschildert also acht Zeilen auf. Diese werden von oben nach unten wie folgt benannt:

D - C - B - A - 8 - 4 - 2 - 1

Die insgesamt erforderlichen 24 Zeilen sind auf der Lochkarte von oben nach unten so untergebracht, wie es die Abb. 1.9. zeigt.

Die D-Zeilen überlappen die dritte Druckzeile, die C-Zeilen die vierte.

Lochungen in den Zeilen D, C, B und A heißen Zonenlochungen; Lochungen in den Zeilen 8, 4, 2 und 1 Ziffernlochungen. Im Anhang A sind alle 256 Verschlüsselungen aufgeführt. Eine gewisse, wenn auch beschränkte, Übereinstimmung mit dem für die 80 spaltigen Lochkarten geltenden Hollerithcode läßt sich finden. Eine 12er Lochung bei einer 80 spaltigen Lochkarte entspricht einer Lochung in den Zeilen A und B, eine 11er Lochung einer Lochung in Zeile B und eine 0er Lochung einer Lochung in Zeile A.

Es gibt Maschinenausrüstungen, die keine Lochkarten akzeptieren, die Lochungen in den Zeilen C und D besitzen. Wird bei einer solchen Ausrüstung der gesamte Zeichenvorrat von 256 Zeichen benötigt, so wird die Lochkarte als eine aus nur 64 Spalten bestehende angesehen. Die Zeilen B und A der Reihe 3 bleiben unbenutzt. Von Reihe 3 werden die Zeilen 8 und 4 als die Zeilen D und C der 1. Reihe angesehen, die Zeilen 2 und 1 als die Zeilen D und C der 2. Reihe.

1.6. Der Kartenlocher und -prüfer IBM 5496

Zum Ablochen der 96 spaltigen Lochkarten dient die IBM 5496. Mit ihr können jedoch nur die 64 Zeichen der bereits mehrfach erwähnten Untermenge abgelocht werden. Die Arbeitsweise entspricht der der IBM 129: Die Datenzeichen wandern in den Speicher. Während aus diesem eine Karte gestanzt und beschriftet wird, kann der Bediener bzw. die Bedienerin bereits die Zeichen für die nächste Karte eintasten. Die Beschriftung der Druckzeilen auf den Karten ist wie folgt geregelt (siehe auch Abb. 1.8. und 1.9.):

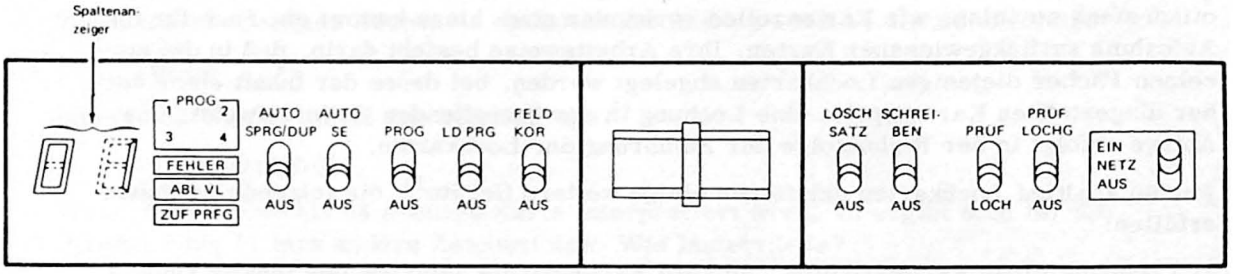
1. Druckzeile —— Zeichen der 1. Reihe
2. Druckzeile —— Zeichen der 2. Reihe
3. Druckzeile —— Zeichen der 3. Reihe

Durch die IBM 5496 wird die 4. Zeile nicht beschriftet.

In der Abb. 1.10. ist das Steuerpult der IBM 5496 dargestellt. Vier Programme sind für das Eintasten von Daten verfügbar; zusätzlich besteht wie bei der IBM 29 die Möglichkeit, ohne Programmkarte einzutasten. Die Taste RECHTS STELLG[1] bei der IBM 5496 entspricht der Wirkungsweise der Taste LEERSPALTEN —— NULL LINKS STRG

1) Abkürzung von „Rechtsbündige Stellung".

bei der IBM 129. Die Taste FELD LÖSCH bei der IBM 5496 kommt der Taste ZEICH RÜCK der IBM 129 gleich. Und schließlich bewirkt bei der IBM 5496 das Anschlagen der Taste LESEN das Lesen einer Datenkarte genau so wie bei der Datenleseoperation bei der IBM 129. Zum Vergleich mit der IBM 129 ist der Abschnitt 1.4. heranzuziehen, insbesonders ist die Abb. 1.7. mit der Abb. 1.10. zu vergleichen.



a) Schalterplatte über der Tastatur

b) Tastatur (deutsche Belegung)

Abb. 1.10. Das Steuerpult der IBM 5496

1.7. Andere Lochkartenmaschinen und -geräte

Mit dem Begriff Satzeinheit [1] werden gewöhnlich solche Maschinen bzw. Geräte charakterisiert, die die herkömmlichen Lochkarten verarbeiten. Die Karte selbst wird als ein Satz angesehen. Da in vielen kommerziellen Anwendungen jeder Satz auf eine separate Lochkarte umgesetzt wird, heißt das, daß die damit verbundene Verarbeitung der Verarbeitung einzelner Karten gleichkommt.

Neben den Lochkartengeräten, die an Computer angeschlossen werden, gibt es Kartenlocher, Kartenprüfer und Kartensortiermaschinen sowohl für 80 spaltige als auch für 96 spaltige Lochkarten. Kartenprüfen bedeutet, daß an einem äußerlich wie ein Locher aussehenden Kartenprüfer das Eintasten der gleichen Zeichen wiederholt wird, die auf den zu prüfenden Karten bereits abgelocht vorhanden sind. Eine Diskrepanz zwischen dem eingetasteten und dem bereits abgelochten Zeichen bewirkt das Einstanzen einer Kerbe oberhalb der fehlerhaften Spalte. Stimmt

1) Englische Bezeichnung ist unit record.

das am Prüfer eingetastete Kartenbild mit der Ablochung der Karte überein, so wird eine Kerbe am rechten Kartenrand plaziert. Sowohl die IBM 129 als auch die IBM 5496 besitzen einen Umschalter, der ihren Einsatz auch als Kartenprüfer ermöglicht. Sie sind also Zweifunktionsgeräte, Kartenlocher und Kartenprüfer zugleich.

Bei Sortiermaschinen handelt es sich um Geräte mit mehreren Ablagefächern, mindestens sovielen, wie Kartenzeilen vorhanden sind; hinzu kommt ein Fach für die Aufnahme zurückgewiesener Karten. Ihre Arbeitsweise besteht darin, daß in die einzelnen Fächer diejenigen Lochkarten abgelegt werden, bei denen der Inhalt einer vorher eingestellten Kartenspalte eine Lochung in der betreffenden Zeile aufweist. Die Ablage erfolgt in der Reihenfolge der Zuführung der Lochkarten.

Für 80 spaltige Lochkarten existieren einige weitere Geräte[1], die folgende Aufgaben erfüllen:

a) Trennung eines Kartenpaketes[2] in zwei Kartenpakete aufgrund des Inhalts ausgewählter Kartenspalten
b) Mischen von zwei sortierten Kartenpaketen zu einem einzigen, ebenfalls sortierten Kartenpaket
c) Beschriften von Karten mit dem Inhalt von ausgewählten Spalten
d) Stanzen der Ergebnisse arithmetischer Operationen mit in den Karten abgelochter Zahlen in die Karten selbst
e) Drucken von in Karten enthaltenen Informationen auf Papier (Tabellieren)

Aufgabengruppe 1

1) Klappt man eine IBM-Lochkarte um, so hat das zur Folge, daß die Zeilen 12 und 9, 11 und 8 usw. vertauscht werden. Wie ist der Inhalt der nachstehenden Karten zu interpretieren, wenn eine solche Umklappung durchgeführt wird?

   a) 3+2=5 b) $36 c) HIGHLY d) ERROR

2) Eine Lochkarte ist wie folgt abgelocht:

   a) 3+2=5 b) $36 c) HIGHLY d) ERROR

   In jede Spalte wird nun zusätzlich noch die Zeile 8 abgelocht. Wie müßte dann der Inhalt der Karte gelesen werden?

3) Auf einem Locher des Typs IBM 29 sind in einer Karte folgende Mehrfachlochungen vorgenommen worden:

   a) 0,3,8, b) 0,5,8, c) 0,8,9, d) 0,1,8, e) 12,11,0,4, f) 2,4,8, g) 12,11,9

   Welche Zeichen werden durch diese Lochkombinationen dargestellt, falls überhaupt welche dargestellt werden?

1) Übersichtliche Kurzbeschreibungen hierzu in Clarence B. Germain, Das Programmier-Handbuch der IBM/360, Carl Hanser Verlag, München 1969.

2) Unter einem Kartenpaket versteht man eine Menge von Lochkarten; in der Praxis vielfach auch als Kartensatz bezeichnet.

4. Mittels Mehrfachlochung werden bei einer 80 spaltigen Lochkarte folgende Lochkombinationen in einer Spalte erzeugt:

   a) 0,3,8 b) 0,5,8 c) 0,8,9 d) 0,1,8 e) -,0,4 f) 2,4,8 g) -,0,9

   Welche Zeichen werden durch diese Lochkombinationen dargestellt?

5. Bei einer 96 spaltigen Lochkarte enthalten die Spalten 1 bis 11 die Zeichenfolge [1)]

   WHENb_23>9'

   und die Spalten 64 bis 75 die Zeichenfolge [1)]

   b444b@b46b@

   Wenn diese Karte als 64 spaltige Karte interpretiert wird, so ergibt sich für die Spalten 1 bis 11 eine andere Zeichenfolge. Wie lautet diese?

6) Wie muß eine Programmkarte für den Locher IBM 29 abgelocht werden, wenn durch das Programm 1 auf allen Spalten nur Alpha-Zeichen gelocht werden sollen, durch das Programm 2 auf allen Spalten nur numerische Zeichen?

7) Es soll eine Programmkarte für den Locher IBM 29 entworfen werden, die folgende Aufgabenstellung erfüllen hilft:

   Auf den Spalten 1 bis 5 ist ein fünfstelliges Feld abzulochen, das führende Nullen (NULL LINKS) berücksichtigt. Auf den nächsten drei Spalten folgt ein dreistelliges numerisches Feld. Daran schließt sich ein 21-stelliges alphanumerisches Feld auf den Spalten 9 bis 29 an. Die Spalten 30 bis 72 sind zu überspringen. Die auf den Spalten 73 bis 80 der vorhergehenden Karte enthaltenen Informationen alphanumerischer Art sind zu duplizieren (auf der ersten Karte des Kartenpaketes sind diese natürlich abzulochen).

1) b bedeutet hier das Leerzeichen (Zwischenraumzeichen).

2. Einführung in mathematische Bezeichnungen und in die mathematische Schreibweise

2.1. Reelle Zahlen

Beim Zählen beginnen wir mit der Zahl 1, gehen dann zur 2, 3 usw. Diese Zahlen bezeichnet man als natürliche Zahlen. Wir sagen, daß die natürlichen Zahlen die Menge 1, 2, 3, 4, ··· bilden.

Wollen wir sowohl vorwärts als auch rückwärts zählen, so stoßen wir auf die Menge der ganzen Zahlen. Beim Vorwärtszählen bekommen wir +1, +2, +3, ··· und beim Rückwärtszählen von 1 an erhalten wir 0, -1, -2, ···. Die natürlichen Zahlen, welche manchmal auch als ganze Zahlen ohne Vorzeichen bezeichnet werden, entsprechen also den positiven ganzen Zahlen. Die ganzen Zahlen können wir in drei Untermengen unterteilen:

a) positive ganze Zahlen,
b) Null,
c) negative ganze Zahlen.

Positive ganze Zahlen werden häufig ohne Vorzeichen geschrieben; so ist es gleichgültig, ob man 789 statt +789 notiert. Da Null weder positiv noch negativ ist, besitzt diese Zahl kein Vorzeichen. Findet man irgendwo einmal die Schreibweise +0 oder -0, so ist diese höchstens aus Bequemlichkeitsgründen erfolgt, in jedem Fall bedeutet sie jedoch nichts weiter als einfach die Zahl 0.

Zeichnen wir uns eine gerade Linie und markieren wir auf ihr einen Punkt mit 0 und einen zweiten rechts davon gelegenen mit +1, so haben wir damit eine Skala festgelegt. Wir können den Abstand von 0 nach 1 als Einheitsstrecke hernehmen und wiederholt nach rechts abtragen, wodurch wir die positiven ganzen Zahlen erhalten; tragen wir sie wiederholt, von der 0 ausgehend, nach links ab, so bekommen wir die negativen ganzen Zahlen. Ein solches Gebilde (siehe Abb. 2.1.) bezeichnen wir als Zahlengerade.

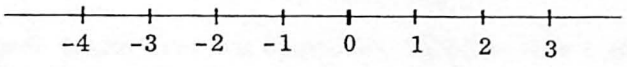


Abb. 2.1. Zahlengerade

Wir haben also bestimmten Punkten der Geraden eine ganze Zahl zugewiesen. Der Abstand des Punktes von dem mit 0 bezeichneten Punkt stellt ihren Wert dar, ihr Vorzeichen wird dadurch bestimmt, ob der betreffende Punkt links (-) oder rechts (+) vom Punkt 0 liegt. Zwischen aufeinanderfolgenden, durch ganze Zahlen charakterisierten Punkten, liegen nun weitere Punkte auf der Geraden, die wir auf die gleiche Weise durch Zahlen bezeichnen können. Alle diese Zahlen bilden die nichtabzählbare Menge der reellen Zahlen; die Menge der natürlichen und die Menge der ganzen Zahlen sind nur Untermengen (Teilmengen) der Menge der reellen Zahlen.

Unter dem absoluten Wert oder absoluten Betrag einer Zahl verstehen wir ihren Wert ohne Berücksichtigung des Vorzeichens. Auf unsere Zeichnung bezogen, drückt sich demnach der absolute Wert einer Zahl als die Entfernung des ihr zugewiesenen Punktes vom Nullpunkt aus. Der absolute Betrag einer Zahl wird wie folgt geschrieben: Man setzt die Zahl zwischen zwei vertikale Striche. Beispiele:

a) $|3,4| = 3,4$ b) $|-5,6| = 5,6$ c) $|0| = 0$ d) $|-4| = 4$ e) $|4| = 4$ f) $|+4| = 4$

Für absolute Werte kann man folgende Regeln niederschreiben:

$$\begin{aligned} |q| &= q \quad \text{für } q > 0 \\ |q| &= 0 \quad \text{für } q = 0 \\ |q| &= -q \quad \text{für } q < 0 \end{aligned}$$

Eine der wichtigsten Eigenschaften reeller Zahlen ist die, daß sie geordnet sind. Für zwei beliebige reelle Zahlen p und q folgt daraus:

a) p ist entweder die gleiche Zahl wie q oder
b) p liegt rechts von q oder
c) p liegt links von q.

Stellt p die gleiche Zahl wie q dar, so sagen wir, daß p gleich q ist und schreiben $p=q$; um anzudeuten, daß q nicht gleich p ist, schreiben wir $p \neq q$. Liegt p rechts von q, so sagen wir, daß p größer als q ist und schreiben dafür $p>q$. Liegt dagegen p links von q, so sagen wir, daß p kleiner als q ist und schreiben dafür $p<q$. Um anzudeuten, daß p nicht größer als q ist, können wir auch schreiben $p \ngtr q$; ebenso können wir schreiben $p \nless q$ und deuten damit an, daß p nicht kleiner als q ist. Einige andere Vergleichssymbole werden ebenfalls noch verwendet. Es handelt sich im einzelnen um folgende:

a) $p \geqq q$ bedeutet, daß entweder $p=q$ ist oder daß $p>q$ ist. Es liegt damit die gleiche Beziehung zwischen p und q vor, wie sie durch die Schreibweise $p \ngtr q$ ausgedrückt werden kann.

b) $p \leqq q$ bedeutet, daß entweder $p=q$ ist oder daß $p<q$ ist. Es besteht damit zwischen p und q die gleiche Beziehung, die wir auch durch die Schreibweise $p \ngtr q$ ausdrücken können.

Zum Schluß wollen wir einige Beispiele für diese Beziehungen betrachten:

$$3=3;\ 3>2;\ 3<4;\ 3 \geqq 1;\ 3 \leqq 8;\ -3<2;\ -3>-4;\ 7 \ngtr 14;\ 7 \leqq 14;\ 0<\frac{1}{2};\ 0 \geqq -\frac{1}{2};\ 4 \neq 5.$$

2.2. Indizes

Um einzelne Mitglieder einer Klasse unterscheiden zu können, muß man sie durch ein besonderes Merkmal kennzeichnen. Man kann das am besten erreichen, indem man an die Symbole kleine tiefergestellte natürliche Zahlen anhängt, die somit auf ein spezielles Mitglied der Klasse verweisen. Liegen beispielsweise drei Gewichtsmessungen eines Gegenstandes vor (34,8 g, 35,2 g und 34,9 g), so können wir die Gewichte symbolisch wie folgt kennzeichnen:

$$g_1 = 34,8,\ g_2 = 35,2 \text{ und } g_3 = 34,9.$$

Wir bezeichnen die 1, die 2 und die 3 als Indizes. Mitunter ist es auch erforderlich, von einer Gewichtsmessung im allgemeinen zu sprechen, ohne auf eine spezielle Gewichtsmessung Bezug nehmen zu müssen. In einem solchen Fall können wir allgemein schreiben g_i und verstehen unter g_i ein allgemeines Symbol für das Gewicht. Unter i müssen wir dann eine variable Größe verstehen, die die Werte 1, 2 oder 3 annehmen kann. Obwohl beliebige Symbole zur Darstellung der Indizes verwendet werden dürfen, hat es sich eingebürgert, daß man für Indizes vor allem die mittleren Buchstaben des Alphabetes benutzt, also meist i, j, k, l, m und n.

Mitunter muß man auch mehr als einen Index verwenden. Die verschiedenen Indizes werden im allgemeinen unmittelbar nebeneinander geschrieben, beispielsweise also x_{13}, x_{km}, x_{ijk}, t_{21}. Sollte daraus jedoch ein Mißverständnis entstehen können, so kann man die Indizes durch Kommas voneinander trennen:

$$x_{1,3}, \quad x_{k,m}, \quad x_{i,j,k}, \quad t_{2,1}.$$

Das Symbol Σ (großer griechischer Buchstabe Sigma) wird zur Bezeichnung von Summen verwendet. Schreibt man z. B.

$$\sum_{i=m}^{i=n} x_i \quad \text{oder kurz} \quad \sum_{i=m}^{n} x_i \quad ,$$

so meint man damit, daß die Summe aller x_i-Werte zu bilden ist, wobei die Indizes von m bis n laufen. Einige Beispiele mögen das erläutern:

$$\sum_{j=2}^{4} x_j = x_2+x_3+x_4$$

$$\sum_{i=1}^{4} (i^2) = 1^2+2^2+3^2+4^2 = 30$$

Beim Auflösen der Summenzeichen ist also wie folgt zu verfahren: Man nimmt den hinter dem Summenzeichen stehenden Ausdruck her und ersetzt die darin auftretende veränderliche Größe i durch den kleinsten Wert von i. Dieser ist unterhalb des Summenzeichens angegeben. Dazu addiert man die Ausdrücke, die man dadurch bildet, daß man i jeweils um 1 erhöht, bis man schließlich ein letztes Glied gefunden hat, bei dem i durch den höchstmöglichen Wert ersetzt ist. Der Buchstabe i wird oft als Summenindex bezeichnet, m heißt die untere Grenze und n die obere Grenze der Summation. Ergeben sich die Summationsgrenzen offensichtlich aus dem textlichen Zusammenhang, so können sie auch weggelassen werden.

Durch $\prod$ (großer griechischer Buchstabe Pi) wird ein Produkt gekennzeichnet; die Verwendung des Zeichens erfolgt genau so wie es beim Summenzeichen erläutert wurde. Bleiben wir bei den obigen Beispielen:

$$\prod_{j=2}^{4} x_j = x_2 \cdot x_3 \cdot x_4$$

$$\prod_{j=1}^{4} (j^2) = 1^2 \cdot 2^2 \cdot 3^2 \cdot 4^2 = 576$$

2.3. Zahlensysteme

Wenn wir die Zahl 327 schreiben, so können wir uns diese in $3 \cdot 10^2+2 \cdot 10^1+7 \cdot 10^0$ zerlegt denken. Ebenso können wir die Zahl 87,75 zu $8 \cdot 10^1+7 \cdot 10^0+7 \cdot 10^{-1}+5 \cdot 10^{-2}$ auffassen. Der Vorteil des Dezimalsystems besteht darin, daß wir beliebig große

Zahlen allein durch Verwendung von nur 10 Ziffern (Zahlzeichen) darstellen können; hinzu kommt, falls notwendig, ein Dezimalpunkt (Dezimalkomma). Geht man vom Dezimalpunkt nach links, so stellt die erste links davor stehende Ziffer ihren Wert direkt dar; um den durch die zweite Ziffer repräsentierten Wert zu finden, muß diese mit der ersten Potenz von 10 multipliziert werden, die nächste Ziffer mit der zweiten Potenz usw. Die erste Ziffer rechts vom Dezimalpunkt stellt den Wert dar, den man erhält, wenn man sie mit 1/10 multipliziert, die zweite Ziffer repräsentiert einen Wert, den man erhält, wenn man sie mit 1/100 multipliziert usw. Da der durch eine Ziffer repräsentierte Wert von der Stellung der Ziffer innerhalb der Zahl abhängt, spricht man beim Dezimalsystem von einem Stellenwertsystem.

Nun besteht aber kein zwingender Grund, daß wir die Zahlen als eine Summe von Gliedern darstellen, die sich auf Zehnerpotenzen beziehen. Die Araber machten sich vermutlich dieses Zahlensystem deshalb zu eigen, weil sie die zehn Finger dazu verlockten; es gibt aber sogar heute noch Völker, die andere Zahlensysteme benutzen.

Betrachten wir z.B. noch einmal die Zahl 327. Wir können nämlich 327 logisch genau so gut als $5\cdot 8^2+0\cdot 8^1+7\cdot 8^0=507_8$ auffassen; der Index 8 soll anzeigen, daß wir es hier mit dem Zahlensystem zur Basis 8 zu tun haben. Zahlen, die im Zahlensystem mit der Basis 8 vorliegen, werden als Oktalzahlen bezeichnet; sie werden ebenfalls vielfach benutzt. Wir können 327_{10} auch als $1\cdot 16^2+4\cdot 16^1+7\cdot 16^0=147_{16}$ auffassen. Zahlen, die im Zahlensystem mit der Basis 16 ausgedrückt werden, heißen Sedezimalzahlen [1]. Für Sedezimalzahlen wollen wir die Abkürzung sed einführen.

Als einfachste Basis für ein Zahlensystem können wir die Zahl 2 ansehen. 327_{10} können wir zerlegen in $1\cdot 2^8+0\cdot 2^7+1\cdot 2^6+0\cdot 2^5+0\cdot 2^4+0\cdot 2^3+1\cdot 2^2+1\cdot 2^1+1\cdot 2^0$, wodurch wir auch schreiben können 101000111_2. Benutzen wir das Zahlensystem zur Basis 2, so bedienen wir uns eines erheblichen Vorteils, brauchen wir nunmehr doch nur noch zwei Ziffern zu benutzen. Nachteilig ist jedoch, daß Dualzahlen eine weitaus größere Stellenzahl aufweisen, als Zahlen in einem Zahlensystem mit einer größeren Basis. Binärzahlen, wie die Dualzahlen oft auch genannt werden, erfordern z.B. etwa 3,3 mal so viele Stellen wie die gleichwertigen Dezimalzahlen.

Im Dezimalsystem werden 10 Ziffern benötigt, Oktalzahlen erfordern acht Ziffern, Dualzahlen zwei Ziffern, Sedezimalzahlen also 16 Ziffern. Uns sind aber nur die 10 Ziffern des Dezimalsystems überliefert worden; wir könnten uns darum bemühen, neue Zahlzeichen festzulegen. Man hat sich aber geeinigt, stattdessen die ersten Buchstaben des Alphabets zu verwenden. So steht A für die den Wert 10_{10} repräsentierende Sedezimalziffer, B für 11_{10}, C für 12_{10}, D für 13_{10}, E für 14_{10} und F für 15_{10}.

Somit bedeutet also die Sedezimalzahl $2E_{16}$ die Dezimalzahl $2\cdot 16^1+14\cdot 16^0=46_{10}$.

Brüche werden in Zahlensystemen, die eine andere Basis als 10 besitzen, genau so wie im Dezimalsystem behandelt. Man muß jedoch sich eines immer vergegenwärtigen: Endliche Brüche eines Systems können durchaus in einem anderen System unendliche Brüche darstellen. Man betrachte folgende zwei Fälle, im letzteren liegt ein Beispiel für den eben geschilderten Sachverhalt vor:

$$0,38_{16}=0,21875_{10}$$
$$0,2_{10}=0,\overline{3}\ldots{}_{16}$$

1) Im amerikanisch-englischen Sprachraum wird das Sedezimalsystem als Hexadezimalsystem bezeichnet; diese Bezeichnung ist auf dem Gebiet der Datenverarbeitung auch im deutschen Sprachraum verbreitet. Als Abkürzung wird allgemein üblich hex benutzt.

Gelegentlich werden wir mit einem bestimmten Problem konfrontiert, nämlich mit der Konvertierung einer Zahl von einem Zahlensystem in ein anderes. Die Konvertierung zwischen dem dualen System und dem sedezimalen System ist einfach: Wir brauchen nur jede Sedezimalziffer als vier Dualziffern aufzufassen. So kann z.B. 147_{16} zu 0001 0100 0111 geschrieben werden, stellt also die Dualzahl 101000111_2 dar. Dieser Zusammenhang ist leicht zu erkennen, da nämlich $2^4=16$ ist. (Um zu unserer allgemein üblichen Schreibweise zurückzukehren, wollen wir an dieser Stelle vereinbaren, daß wir den Index bei Dezimalzahlen nur noch dann niederschreiben wollen, wenn die Gefahr einer Verwechslung besteht).

Eine ähnliche Methode wie bei der Dual-Sedezimal-Konvertierung können wir bei der Dual-Oktal-Konvertierung anwenden, da $2^3=8$ ist. Dagegen bereitet die Konvertierung vom Dezimalsystem ins Sedezimalsystem (oder ins Dualsystem oder ins Oktalsystem) und umgekehrt einige Schwierigkeiten. Um sich mit der allgemeinen Methode vertraut zu machen, genügt es, wenn wir vier verschiedene Beispiele betrachten [1].

Beispiele

1) Es ist die Dezimalzahl 7143 in die entsprechende Sedezimalzahl zu konvertieren!

1. Schritt: 7143:16=446 Rest 7
2. Schritt: 446:16= 27 Rest 14 ($\hat{=}$ Rest E)
3. Schritt: 27:16= 1 Rest 11 ($\hat{=}$ Rest B)
4. Schritt: 1:16= 0 Rest 1

Daraus ergibt sich, daß $7143_{10}=1BE7_{16}$ ist.

2) Es ist die Dezimalzahl 0,2175 in die entsprechende Sedezimalzahl zu konvertieren!

1. Schritt: $0,2175\cdot 16= 3,48=3+0,48$
2. Schritt: $0,4800\cdot 16= 7,68=7+0,68$
3. Schritt: $0,6800\cdot 16=10,88=A+0,88$
4. Schritt: $0,8800\cdot 16=14,08=E+0,08$
5. Schritt: $0,0800\cdot 16= 1,28=1+0,28$
6. Schritt: $0,2800\cdot 16= 4,48=4+0,48$
7. Schritt: $0,4800\cdot 16= 6,68=7+0,68$

Das Ergebnis des 7. Schrittes ist also gleich dem Ergebnis des 2. Schrittes, d.h. aber, daß die entwickelte Sedezimalzahl periodisch ist.

Es ist also $0,2175_{10}=0,3\overline{7AE14}\ldots$

3) Die Sedezimalzahl $9F3A_{16}$ ist in die entsprechende Dezimalzahl zu konvertieren!

1. Schritt: $9\cdot 16= 144$
2. Schritt: $144+F= 159$
3. Schritt: $159\cdot 16= 2544$
4. Schritt: $2544+3= 2547$
5. Schritt: $2547\cdot 16=40752$
6. Schritt: $40752+A=40762$

Somit ist also $9F3A_{16}=40762_{10}$

1) Näheres siehe hierzu: Rüdiger und Anna Gritsch: Das Programmieren von Computern. Carl Hanser Verlag, München 1972. ISBN 3-446-11519-6.

4) Die Sedezimalzahl $0,9B28_{16}$ ist in die entsprechende Dezimalzahl zu konvertieren!

1. Schritt: $8:16=0,5$
2. Schritt: $0,5+2=2,5$
3. Schritt: $2,5:16=0,15625$
4. Schritt: $0,15625+B=11,15625$
5. Schritt: $11,15625:16=0,697265625$
6. Schritt: $0,697265625+9=9,697265625$
7. Schritt: $9,697265625:16=0,6060791015625$

Somit ist $0,9B28_{16}=0,6060791015625_{10}$

Bei diesem Beispiel ist es bei Benutzung eines Tischrechners bequemer, wenn man mit $\frac{1}{16} = 0,0625$ multipliziert, statt daß man durch 16 dividiert.

Aufgabengruppe 2

1) Aus der nachstehend aufgeführten Zahlenmenge sind folgende Zahlen auszuwählen:

a) alle positiven ganzen Zahlen,
b) alle Zahlen, deren absoluter Wert kleiner als 2 ist,
c) alle Zahlen, die nicht größer als -3 sind,
d) alle Zahlen, die größer oder gleich 4 sind!

Die Zahlen lauten: $-2; -2,6; -7/3; 0; \sqrt{7}; 670/5$.

2) Berechne folgende Summen:

a) $\sum_{i=1}^{3} (i^2-3i)$

b) $\sum_{j=2}^{5} (j-3)^2$

c) $\sum_{k=3}^{5} \left(\sum_{i=1}^{2} k^i \right)^2$

d) $\sum_{n=3}^{45} 7$

3) Gegeben sind: $x_1=3$, $x_2=4$, $x_3=-3$, $x_4=2$ und $x_5=1$. Gesucht sind folgende Werte:

a) Σx b) Σx^2 c) $\Sigma (x-1,2)^2$ d) Πx e) $\Pi (x-3,4)$

4) Konvertiere die folgenden Dezimalzahlen in Dualzahlen!

a) 7 b) 26 c) -6 d) 6,375 e) -0,00304

5) Konvertiere die folgenden Dualzahlen in Dezimalzahlen!

a) 101 101 b) 111 001 c) 10 101 000 d) 11,11 e) 0,001 101

6) Konvertiere jede der Zahlen in Aufgabe 5) in Sedezimalzahlen!

7) Konvertiere die folgenden Dezimalzahlen in die entsprechenden Sedezimalzahlen!

a) 32 769 b) -11200 c) 16 389 195 d) 77,875 e) 1,39 f) 2/7 g) 555,441
h) 123 456 789, 012 345

8) Konvertiere die folgenden Sedezimalzahlen in Dezimalzahlen!

a) 12A 3D4 b) 8BC c) 0,18 d) 45,55 e) 32 769 f) 123 456, 789 ABF
g) FF FFE h) 3EC BFA 001, CBC

Anmerkung: Der besseren Lesbarkeit wegen sind bei den einzelnen Zahlen die Ziffern zu Gruppen von je drei Ziffern zusammengefaßt.

2.4. Die Gleitpunktdarstellung[1)]

Eines der ernsthaftesten Probleme bei verwickelten Berechnungen besteht darin, die Stellung des Dezimalpunktes genau zu verfolgen. Bei vielen kaufmännischen Berechnungen ist das nicht allzu schwierig; wenn man z.B. mit DM und Pfg. rechnet, so weiß man mit Bestimmtheit, wo der Dezimalpunkt stehen muß. Bei mathematischen und technischen Berechnungen jedoch kann die Stellung des Dezimalpunktes erheblich differieren; oft kann sogar der Mathematiker die Stellung des Dezimalpunktes beim Ergebnis nicht ohne weiteres voraussagen, wenn bestimmte Berechnungen auszuführen sind.

Wir wollen diese Schwierigkeiten an einem relativ einfachen Problem verfolgen. Es soll die Kurve der Funktion

$$y = \frac{x + 11}{8x^2 - 26x + 21}$$

gezeichnet werden.

Zunächst stellen wir eine Wertetafel auf, indem wir willkürlich Werte von x vorgeben und die dazugehörigen y-Werte berechnen. Auf diese Weise finden wir folgende Tabelle:

x	y
-20	-0,00241
-10	+0,000925
0	+0,524
1,2	+9,24
1,4	+44,3
1,6	-105,0
1,8	+107,0
2	+13,0
10	+0,0374
20	+0,0115

Wir erkennen, daß die y-Werte über einen weiten Wertebereich schwanken; dividieren wir den größten Wert aus der Tabelle durch den kleinsten, so erhalten wir einen Faktor, der größer als 10^5 ist. Wählen wir noch weitere x-Werte aus und nehmen sie in die Tabelle auf, so werden wir feststellen, daß der Unterschiedsfaktor weit über 10^5 ansteigt.

1) Statt Gleitpunktdarstellung sagt man auch Gleitkommadarstellung.

Um das Problem zur Bestimmung der Stellung des Dezimalpunktes in den Griff zu bekommen, ist es zweckmäßig, jede reelle Zahl als Produkt aus einem Dezimalbruch zwischen 0,1 und 1 und einer Potenz von 10 darzustellen.

Beispiele:

$$0,0028 = 0,28 \cdot 10^{-2}$$
$$108,7500 = 0,10875 \cdot 10^{3}$$

Zahlen, die in dieser Form niedergeschrieben werden, bezeichnet man als Zahlen in Gleitpunktdarstellung. Schreibt man eine Zahl als Dezimalbruch, multipliziert mit einer geeigneten Potenz von 10, und ist dieser Dezimalbruch kleiner als 0, 1, beispielsweise also $2,3333 = 0,023333 \cdot 10^{2}$, so spricht man hierbei von einer nicht normalisierten Gleitpunktdarstellung.

Der gebrochene Anteil einer Zahl in Gleitpunktdarstellung, kurz auch als Gleitpunktzahl bezeichnet, heißt Mantisse, der Exponent der Zehnerpotenz heißt Charakteristik. (Diese Bezeichnungen gelten selbstverständlich auch, wenn man der Gleitpunktdarstellung ein anderes Zahlensystem und nicht das Dezimalsystem zugrundelegt). Da keine Möglichkeit besteht, Exponenten in Lochkarten zu lochen, werden die Zahlen häufig in der sogenannten Exponentenform abgelocht; das bedeutet, daß einer Dezimalzahl der Buchstabe E (unter gewissen Voraussetzungen manchmal auch D) folgt, dem sich wiederum der Exponent anschließt. Dieser muß ganzzahlig sein. Um das zu veranschaulichen, wollen wir einige mögliche Formen an zwei Beispielen verfolgen:

a) $0,28 \cdot 10^{-2}$ = 0.28E-2
= .28E-2
= 2.8E-3
= 28.0E-04

b) 108,75 = 0.10875E+3
= 0.10875E3
= .10875E3

Wird die Gleitpunktdarstellung auf Sedezimalzahlen angewendet, so werden diese selbstverständlich als Produkt eines Bruches und einer Potenz von 16 ausgedrückt. Die Gleitpunktdarstellung ist normalisiert, wenn der Bruch zwischen $0,1_{16}$ und 1_{16} liegt. Den Wert von $0,1_{16}$, dezimal ausgedrückt, findet man, daran sei noch einmal erinnert, indem man 1 durch 16 dividiert.

Theoretisch ist die Anzahl der Stellen nach dem Komma bei reellen Zahlen nicht begrenzt. Praktisch müssen wir die Rechnungen jedoch auf eine bestimmte Stellenzahl nach dem Komma begrenzen, oft sogar auf eine sehr kleine. Die Erfahrung hat gezeigt, daß für die meisten praktischen Berechnungen Mantissenlängen von 8 (bei Dezimalzahlen) bzw. von 6 (bei Sedezimalzahlen) ausreichen.

2.5. Boolesche Arithmetik

Eine Boolesche Variable kann nur zwei Werte annehmen, die als „wahr" und „falsch" bezeichnet werden. Dargestellt werden diese Werte oft durch 1 (für wahr) und durch 0 (für falsch). Für die Booleschen Variablen sind vier Operatoren festgelegt:

¬ oder ~ bedeutet das logische NICHT (NOT)
| oder ∨ bedeutet das logische ODER (OR)
& oder ∧ bedeutet das logische UND (AND)
⊻ oder ⊕ bedeutet das logische EXKLUSIVE ODER (EXCLUSIVE OR)

Die einzelnen Operationen sind wie folgt definiert:

a) NOT: $\neg 1 = 0$; $\neg 0 = 1$

b) OR: $a \mid b = 1$, außer wenn a und b nicht gleichzeitig 0 sind
$a \mid b = 0$, wenn a=b=0 ist

c) AND: $a \& b = 1$, wenn sowohl a=1 als auch b=1 ist
$a \& b = 0$, wenn entweder a=0 ist oder b=0 ist oder wenn a=b=0 ist

d) EXCLUSIVE OR: $a \veebar b = 1$, wenn $a \neq b$ ist
$a \veebar b = 0$, wenn a = b ist.

Die Booleschen Operationen, auch als logische Operationen bekannt, können auf jede Dualzahl angewendet werden. Werden sie mit Dualzahlen durchgeführt, so werden die logischen Operationen grundsätzlich auf jedes Paar einander entsprechender Dualziffern angewendet. Vier Beispiele sollen das erläutern:

	1100	1100	1100
$\sim$ 0101	$\vee$0101	$\wedge$0101	$\oplus$0101
1010	1101	0100	1001

Die logische Operation NOT kann nur auf einen Operanden angewendet werden, in die anderen Operationen gehen zwei Operanden ein. Wie für die Addition und für die Multiplikation gilt für OR, AND und EXCLUSIVE OR das kommutative Gesetz, d. h.

$$a \vee b = b \vee a \qquad a \wedge b = b \wedge a \qquad a \oplus b = b \oplus a$$

Die Reihenfolge der Operationen, die in einem Ausdruck ausgeführt werden soll, kann durch Klammern beeinflußt werden.

$$1 \vee (0 \wedge 1) = 1 \vee 0 = 1$$
$$\neg (1 \wedge (1 \vee 1)) = \neg (1 \wedge 1) = \neg 1 = 0$$

Werden keine Klammern verwendet, so gilt folgende Rangordnung für die logischen Operatoren:

Stufe	Operation
1	NOT
2	AND
3	OR, EXCLUSIVE OR

Als höchste Rangordnung gilt hierbei die Stufe 1. Einige Beispiele für die durch diese Rangordnung bedingte Auflösung logischer Ausdrücke sollen diese Betrachtungen abschließen:

a) $1 \vee 0 \wedge 1 = 1 \vee 0 = 1$

b) $1 \wedge 1 \vee 1 = 0 \wedge 1 \vee 1 = 0 \vee 1 = 1$

In diesem Buch wollen wir grundsätzlich die Schreibweise verwenden, die in der Literatur über die Systeme IBM /360 und IBM /370 benutzt wird, nämlich $\neg$, |, & und $\oplus$.

2.6. Komplementierung

Unter dem Komplement von 10 (Zehnerkomplement) einer ganzen Dezimalzahl verstehen wir das Resultat, das sich ergibt, wenn wir diese Zahl von der ganzen Zahl subtrahieren, die wie folgt aufgebaut ist: Sie beginnt mit der Ziffer 1, dieser werden soviele Nullen angehängt wie die zu komplementierende Zahl Ziffern aufweist. Somit berechnet sich das Komplement von 34974 wie folgt:

```
  100000
- 34974
  65026
```

Aus diesem einen Beispiel bereits kann man auf folgende Regel zur Bildung des Zehnerkomplementes schließen: Man subtrahiert jede Ziffer der zu komplementierenden Zahl von 9 und addiert zum so gefundenen Ergebnis eine 1.

Gewöhnlich werden Komplementärzahlen zur Darstellung von negativen Zahlen verwendet. Bei fast allen Tischrechenmaschinen werden negative Zahlen durch ihr Zehnerkomplement dargestellt. Daraus ergibt sich als wesentlicher Vorteil, daß die Additionen sehr vereinfacht werden. Um beispielsweise 8976 zu -384 zu addieren, würde dann folgende Operation ablaufen:

```
  00008976
+ 99999616
  00008592
```

Der links stehende Zehnerübertrag geht verloren, wenn angenommen wird, daß der dieser Rechnung zugrundegelegte Tischrechner nur acht Stellen besitzt. Die Subtraktion wird durchgeführt, indem die Komplementärzahl des Subtrahenden gebildet und diese dann anschließend zum Minuenden addiert wird.

Unter dem Neunerkomplement einer ganzen Dezimalzahl verstehen wir die Zahl, die wir erhalten, wenn wir jede Ziffer der gegebenen Zahl von 9 subtrahieren. Um z.B. das Neunerkomplement von 34974 zu erhalten, müssen wir folgende Rechenoperation ausführen:

```
  99999
- 34974
  65025
```

Wir erkennen aus diesem Beispiel sehr schön, daß das Zehnerkomplement einer Zahl stets um 1 größer ist als das Neunerkomplement der gleichen Zahl. Der Hauptzweck bei der Bildung von Neunerkomplementen besteht deshalb darin, die Bildung des Zehnerkomplementes auf diese recht einfache Art zu ermöglichen.

Wenn wir uns mit ganzen Dualzahlen beschäftigen, so besitzt das Zweierkomplement die gleiche Bedeutung wie das Zehnerkomplement von ganzen Dezimalzahlen. Dem Einerkomplement bei Dualzahlen entspricht infolgedessen das Neunerkomplement von Dezimalzahlen. Die Bildung des Zweierkomplements der Zahl 10011010 geschieht also wie folgt:

```
  100000000
-  10011010
   1100110
```

Das Einerkomplement der gleichen Zahl wird wie folgt gebildet:

$$\begin{array}{r} 11111111 \\ - \quad 10011010 \\ \hline 01100101 \end{array}$$

Addiert man hierauf eine duale 1, so erhält man das oben errechnete Zweierkomplement auf einem anderen, sicher viel einfacheren Weg.

2.7. Rundungsfehler

Es ist leicht nachzuweisen, daß x=8 und y=-3 die Lösungen des Gleichungssystems

$$\begin{aligned} &\text{I.} \quad 5x + 13y = 1 \\ &\text{II.} \quad 7x + 18y = 2 \end{aligned}$$

sind. Wir können dieses Gleichungssystem am einfachsten durch Anwendung des Additionsverfahrens auflösen. Um dieses Verfahren anwenden zu können, multiplizieren wir zunächst die erste Gleichung mit 7 und die zweite mit -5, so daß die beiden Gleichungen in die nachstehenden Gleichungen übergehen:

$$\begin{array}{rl} \text{I.} & 35x + 91y = \quad 7 \\ \underline{\text{II.}} & \underline{-35x - 90y = -10} \\ \text{I.} + \text{II.} & \qquad y = -3 \end{array}$$

Durch Addition der beiden Gleichungen haben wir also die Lösung für y gefunden.

Statt die beiden Ausgangsgleichungen mit bestimmten Faktoren zu multiplizieren und anschließend zu addieren, können wir sie auch zunächst durch bestimmte Zahlen dividieren, z.B. die erste Gleichung durch 5 und die zweite durch -7. Wir erhalten dann:

$$\begin{array}{rl} \text{I.} & x + \frac{13}{5} y = \frac{1}{5} \\ \underline{\text{II.}} & \underline{-x - \frac{18}{7} y = -\frac{2}{7}} \\ \text{I.} + \text{II.} & \qquad \frac{1}{35} y = -\frac{3}{35} \\ & \qquad\quad y = -3 \end{array}$$

Nehmen wir nunmehr an, wir wären gezwungen, alle Rechenoperationen mit Zahlen in der Gleitpunktdarstellung durchzuführen und es ständen uns dafür nur sieben Mantissenstellen zur Verfügung, so würde das bedeuten, daß kein Zwischenergebnis oder gar das Endergebnis mehr als sieben gültige Ziffern aufweisen kann. Verfolgen wir unseren Rechengang unter diesen Voraussetzungen:

$$\begin{array}{rl} \text{I.} & 1,000\,000\,x + 2,600\,000\,y = \quad 0,200\,000\,0 \\ \underline{\text{II.}} & \underline{-1,000\,000\,x - 2,571\,429\,y = -0,285\,714\,3} \\ \text{I.}+\text{II.} & \qquad 0,028\,571\,y = -0,085\,714\,3 \end{array}$$

Dividieren wir dieses Zwischenergebnis durch den Beifaktor von y, so erhalten wir schließlich folgendes Endergebnis:

$$y = -3,000046$$

Dieses Ergebnis weicht in den letzten beiden Stellen vom erwarteten Endergebnis ab. Dieser Fehler entstand dadurch, daß wir zwei Zahlen subtrahieren mußten, die nahe beieinander liegen, wodurch der Beifaktor von y nur mit fünf gültigen Ziffern bestimmt werden konnte, obwohl wir während der gesamten bis dahin erfolgenden Rechengänge sieben Stellen mitgeführt hatten.

Es sollte klargestellt werden, daß der Rundungsfehler zum großen Teil von den in eine Rechnung eingehenden Zahlen abhängt, aber auch vom ausgewählten Lösungsverfahren. Dieses Thema ist sicherlich geeignet genug, um in gründlichen numerischen Studien untersucht zu werden. Jeder, der bei seinen Rechnungen Näherungswerte benutzen muß, sollte sich jedoch dessen bewußt sein, daß seine Ergebnisse mit Rundungsfehlern behaftet sind. Bei umfangreichen numerischen Berechnungen können sich diese sogar zu so großen Werten aufschaukeln, daß alle gültigen Ziffern verloren gehen, in unserem Fall also sieben. Die Endergebnisse sind damit völlig unbrauchbar.

Aufgabengruppe 3

1) Schreibe in normalisierter Gleitpunktdarstellung!

   a) 23,456 b) 1,0 c) 258,375 d) 0,00057 e) 1,000 f) 0,01875

2) Schreibe jede Zahl der Aufgabe 1) in sedezimaler Gleitpunktdarstellung!

3) Schreibe jede Zahl der Aufgabe 1) in der Exponentenform!

4) Berechne folgende Ausdrücke:

   a) $\neg A \vee B \wedge C$
   b) $A \vee B \wedge C$
   c) $A \vee \neg A \wedge B \wedge C$
   d) $\neg (A \wedge B) \vee C$

   Für A , B und C sind folgende Werte vorgegeben:

   (a) A=1, B=0, C=1; (b) A=0, B=0, C=0; (c) A=1, B=1, C=1; (d) A=0, B=1, C=1.

5) Bestimme das Zehnerkomplement der folgenden Dezimalzahlen:

   a) 677 904 b) 990 090 c) 053 150 d) 000 082

6) Bestimme das Neunerkomplement der Zahlen von Aufgabe 5)!

7) Bestimme das Einerkomplement der folgenden Sedezimalzahlen und drücke das Resultat wieder als Sedezimalzahl aus!

   a) 45 B74 340 b) 43 6DC 0FF c) 00 000 000 d) 1E A00 F12

8) Bestimme das Zweierkomplement für die gleichen Zahlen wie in Aufgabe 7)!

9) Bestimme die Summe der Zahlen von Aufgabe 7a) und Aufgabe 7b)!

10) Löse das Gleichungssystem

$$\begin{aligned} \text{I.}\quad & 233x + 259y = 51 \\ \text{II.}\quad & 9x + 10y = 2 \end{aligned}$$

nach allen im Abschnitt 2.7. beschriebenen Methoden!

11) Errechne das Produkt der beiden Sedezimalzahlen 1E4 und C6!

2.8. Statistik

Oft sehen wir uns solchen Unmengen von Daten gegenübergestellt, daß wir einfach nicht in der Lage sind, sie zu überblicken. Über große Datenmengen gewinnen wir die beste Übersicht, wenn wir sie im Zusammenhang sehen und ihre Eigenschaften beschreiben. Im allgemeinen werden diese Eigenschaften am besten mit dem Mittelwert, der Varianz, der Standardabweichung und dem mittleren Fehler erfaßt.

Der Mittelwert einer Datenmenge $x_1, x_2, \cdots, x_n$ wird mit $\bar{X}$ (Gelesen: X quer) bezeichnet und ist definiert zu

$$\bar{X} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Aus Bequemlichkeit läßt man gewöhnlich den Summationsindex weg und schreibt einfach

$$\bar{X} = \frac{1}{n} \Sigma x$$

Nehmen wir an, daß bei einer Prüfung die folgenden Punktzahlen erreicht wurden: $x_1=78$, $x_2=64$, $x_3=83$, $x_4=91$ und $x_5=29$. Dann ergibt sich $\bar{X}$ zu:

$$\bar{X} = \frac{78 \times 64 + 83 + 91 + 29}{5} = 69,00$$

Der Mittelwert ist also ein Maß für den Durchschnittswert der Daten. Haben wir den Mittelwert festgestellt, so legen wir uns die Frage vor, wie die Daten, vom Mittelwert aus gesehen, streuen. Meist wird dazu die Varianz gebildet; im allgemeinen ist sie auch das beste Maß für die Streuung der Daten. Die Varianz s^2 ist wie folgt definiert:

$$s^2 = \frac{\Sigma (x_i - \bar{X})^2}{n - 1} = \frac{n \Sigma x^2 - (\Sigma x)^2}{n (n - 1)}$$

Die zweite Form erhält man aus der ersten durch eine einfache algebraische Umformung. Für das praktische Rechnen ist die zweite Form sicher am bequemsten anzuwenden. Für die oben gegebenen Punktzahlen ergeben sich folgende Werte:

$$\Sigma x^2 = 26191, \quad \Sigma x = 345, \quad n = 5$$

Daraus ermittelt sich die Varianz zu:

$$s^2 = 596,50$$

Die Standardabweichung s, auch mittlere Abweichung oder Streuung genannt, ist definiert als die Quadratwurzel aus der Varianz. Bei unseren Prüfungspunktzahlen ergibt sich s zu

$$s = \sqrt{s^2} = 24,42$$

Der mittlere Fehler

$$M = \frac{\Sigma \mid x - \bar{X} \mid}{n}$$

ist ein anderes Maß für die Streuung von Daten. Er wird hauptsächlich bei Berechnungen zur Lagerbestandskontrolle und zur Qualitätskontrolle in Anwendung gebracht, da er dort leichter zu berechnen ist als die Varianz. In den meisten anderen Fällen kann jedoch die Varianz einfacher berechnet werden und da außerdem für die meisten Datenarten die Varianz ein zuverlässigeres Maß für die Streuung ergibt, wird der mittlere Fehler recht selten verwendet. Für unsere Prüfungspunktzahlen ergibt sich

$$M = \frac{9 + 5 + 14 + 22 + 40}{5} = 18,00$$

Folgen die Daten der Gaußschen Verteilung, wie das meist der Fall ist, so bekommen wir für M den Näherungswert

$$M = s\sqrt{\frac{2}{\pi}}$$

Mitunter werden die Daten auch in Klassen eingeteilt. Hierzu ein Beispiel:

Klasse x	Anzahl f_i
12 bis 14	1
15 bis 17	4
18 bis 20	7
21 bis 23	3
Gesamtzahl	15

In diesem Fall haben wir unsere Definitionen geringfügig abzuändern:

$$\bar{X} = \frac{\Sigma x_i \cdot f_i}{n} \quad \text{mit} \quad n = \Sigma f_i$$

$$s^2 = \frac{\Sigma (x - \bar{X})^2 f}{n-1} = \frac{n\Sigma(x^2 f) - (\Sigma x_i f_i)^2}{n(n-1)}$$

$$M = \frac{\Sigma \mid x - \bar{X} \mid f}{n}$$

Für die x_i haben wir den Durchschnittswert aus dem Minimalwert und dem Maximalwert jeder Klasse einzusetzen. In unserem Beispiel sind

$$n = 15\ ;\quad X = \frac{13 \cdot 1 + 16 \cdot 4 + 19 \cdot 7 + 22 \cdot 3}{15} = 18{,}40\ ;$$

$$\Sigma\, x^2 f = 13^2 \cdot 1 + 16^2 \cdot 4 + 19^2 \cdot 7 + 22^2 \cdot 3 = 5172\ ;$$

$$s^2 = \frac{15 \cdot 5172 - 276^2}{15 \cdot 14} = 6{,}69\ ;$$

$$M = \frac{5{,}4 \cdot 1 + 2{,}4 \cdot 4 + 0{,}6 \cdot 7 + 3{,}6 \cdot 3}{15} = 2{,}00\ ;$$

Oft werden uns Meßergebnisse vorgelegt, die sich aus Paaren von vermutlich aufeinander bezogenen Größen zusammensetzen. Würden wir beispielsweise mit dem Mathematikzensuren einer Oberklasse vertraut gemacht, erhielten wir aber auch gleichzeitig die Zensuren der Schüler dieser Klasse, die sie bei der Aufnahmeprüfung im Fach Mathematik bekommen haben, so würden wir uns in einem solchen Fall sicherlich die Frage vorlegen, ob oder ob nicht eine lineare Beziehung zwischen diesen beiden Zensuren besteht, und, wenn sie besteht, wie sie lautet. In welchem Maß eine lineare Beziehung zwischen x- und y-Werten besteht, können wir am besten beurteilen, wenn wir den Korrelationskoeffizienten r bilden

$$r = \frac{\Sigma\,(x - \overline{X})(y - \overline{Y})}{\sqrt{\Sigma\,(x - \overline{X})^2 \cdot \Sigma\,(y - \overline{Y})^2}}$$

$$r = \frac{n\Sigma xy - (\Sigma x)(\Sigma y)}{\sqrt{[n\Sigma x^2 - (\Sigma x)^2][n\Sigma y^2 - (\Sigma y)^2]}}$$

Die zweite Form erhalten wir aus der ersten durch algebraische Umformungen. Das Vorzeichen von r hat keine Bedeutung.

Will man die geeignetste gerade Linie zu erhalten, die die Beziehungen zwischen x und y am besten widerspiegelt, so besagt die Methode der kleinsten Quadrate, daß wir für a und b folgende Berechnungen durchzuführen haben, wenn wir als Geradengleichung $y = a + bx$ ansetzen:

$$b = \frac{\Sigma\,(x - \overline{X})(y - \overline{Y})}{\Sigma\,(x - \overline{X})^2} = \frac{n\Sigma xy - (\Sigma x)(\Sigma y)}{n\Sigma\,(x^2) - (\Sigma x)^2}$$

$$a = \overline{Y} - b\overline{X}$$

Zunächst werden wir b berechnen und danach a. Der Koeffizient b des linearen Gliedes der Geradengleichung wird Regressionskoeffizient genannt

2.9. Matrizen

Uns sei die Aufgabe gestellt, das lineare Gleichungssystem

I. $4x - 3y = 11$
II. $6x + 8y = 9$

aufzulösen. Ferner sollen wir auch das lineare Gleichungssystem

I. $4w - 3z = 11$
II. $6w + 8z = 9$

lösen. Wir erkennen auf den ersten Blick, daß beide Probleme äquivalent sind. Um beide Gleichungssysteme zu lösen, brauchen wir uns nur die Koeffizienten und die absoluten Glieder herauszuziehen. Wir können sie in der Form

$$\begin{bmatrix} 4 & -3 & 11 \\ 6 & 8 & 9 \end{bmatrix}$$

niederschreiben. Jede derartige rechteckige Anordnung von Zahlen heißt **Matrix**. Ist die Zahl ihrer Zeilen gleich der Zahl ihrer Spalten, so handelt es sich um eine **quadratische Matrix**; andernfalls spricht man von **rechteckigen Matrizen**. Im obigen Beispiel haben wir es mit einer 2×3 Matrix zu tun, weil sie aus 2 Zeilen und 3 Spalten besteht. Weist eine Matrix nur eine Zeile auf, so spricht man oft von einem **Spaltenvektor**; weist sie dagegen nur eine Spalte auf, so spricht man oft von einem **Zeilenvektor**.

Allgemein wird eine Matrix wie folgt geschrieben:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

In diesem Fall handelt es sich um eine m×n-Matrix. Zur Vereinfachung wollen wir mitunter auch

$$A = [a_{ij}] \quad \text{oder kurz} \quad A = [a]$$

schreiben.

Multiplizieren wir eine Matrix mit einem **Skalar**, d.h. mit einer gewöhnlichen Zahl, so müssen wir jedes **Matrixelement** mit dem Skalar multiplizieren. Beispiel:

$$\text{Ist} \qquad A = \begin{bmatrix} 4 & -3 & 11 \\ 7 & 5 & 9 \end{bmatrix},$$

dann ergibt sich $-2 \cdot A$ zu

$$-2 \cdot A = \begin{bmatrix} -8 & 6 & -22 \\ -14 & -10 & -18 \end{bmatrix}$$

Addition und Subtraktion von Matrizen gleichen Umfanges, d.h. mit gleicher Zeilen- und Spaltenzahl, ist definiert als Addition bzw. Subtraktion entsprechender Elemente der Matrizen, d.h. der Zahlen, die an den gleichen Stellen bei beiden Matrizen stehen. Somit gilt allgemein:

$$\text{Aus} \quad A = [a_{ij}] \quad \text{und} \quad B = [b_{ij}]$$
$$\text{folgt} \quad A + B = [a_{ij} + b_{ij}]$$

An einem Beispiel soll das Vorgehen noch einmal erläutert werden:

$$\text{Es ist} \quad A = \begin{bmatrix} 4 & -3 & 11 \\ 7 & 5 & 9 \end{bmatrix} \quad \text{und} \quad B = \begin{bmatrix} 3 & -1 & -6 \\ 2 & -1 & -4 \end{bmatrix}$$

$$\text{Dann ergibt sich} \quad A + B = \begin{bmatrix} 7 & -4 & 5 \\ 9 & 4 & 5 \end{bmatrix} \quad \text{und}$$

$$A - B = \begin{bmatrix} 1 & -2 & 17 \\ 5 & -6 & 13 \end{bmatrix}$$

Für Matrizen unterschiedlichen Umfanges sind Addition und Subtraktion nicht definiert.

Die Multiplikation zweier Matrizen ist wie folgt definiert:

$$\text{Aus} \quad A = \left[a_{ij} \right] \quad \text{und} \quad B = \left[b_{ij} \right]$$

$$\text{folgt} \quad A \cdot B = \left[\sum_k a_{ik} \cdot b_{kj} \right]$$

Das Matrizenprodukt ist auch für bestimmte rechteckige Matrizen definiert. Ist z. B. A eine mxn-Matrix, B eine nxp-Matrix, dann ist das Produkt eine mxp-Matrix. Ist die Anzahl der Spalten von A jedoch nicht gleich der Anzahl der Zeilen von B, dann ist $A \cdot B$ nicht definiert. An einem Beispiel soll nunmehr die Matrizenmultiplikation demonstriert werden:

$$\text{Es sei} \quad A = \begin{bmatrix} 4 & -3 & 11 \\ 7 & 5 & 9 \end{bmatrix} \quad \text{und} \quad D = \begin{bmatrix} 6 \\ 2 \\ -1 \end{bmatrix}$$

A ist also eine 2x3-Matrix und D eine 3x1-Matrix. Als Produkt ergibt sich dann eine 2x1-Matrix.

$$A \cdot D = \begin{bmatrix} 4 \cdot 6 - 3 \cdot 2 + 11(-1) \\ 7 \cdot 6 + 5 \cdot 2 + 9 \cdot 1 \end{bmatrix} = \begin{bmatrix} 7 \\ 43 \end{bmatrix}$$

Wir wollen nun die Matrizen

$$C = \begin{bmatrix} 4 & -3 \\ 6 & 8 \end{bmatrix}, \quad X = \begin{bmatrix} x \\ y \end{bmatrix} \text{ und } K = \begin{bmatrix} 11 \\ 9 \end{bmatrix}$$

betrachten. Bilden wir

$$C \cdot X = K \quad ,$$

so erhalten wir die Gleichungen, von denen wir am Anfang dieses Abschnittes ausgegangen sind, diesmal in Form von Matrizen geschrieben.

Die transponierte Matrix einer Matrix wird erhalten, wenn man die Zeilen mit den Spalten vertauscht. Die transponierte Matrix von C ist also:

$$C' = \begin{bmatrix} 4 & 6 \\ -3 & 8 \end{bmatrix}$$

Die transponierte Matrix von einer mxn-Matrix ist demnach eine nxm-Matrix.

Eine quadratische Matrix, deren Hauptdiagonalelemente sämtlich gleich 1, deren übrige Elemente aber gleich 0 sind, wird mit E bezeichnet und heißt Einheitsmatrix. So ist z. B. die zweireihige Einheitsmatrix gleich

$$E = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

und die dreireihige Einheitsmatrix gleich

$$E = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Wird die Einheitsmatrix mit einer beliebigen anderen Matrix multipliziert, so erhält man als Produkt die andere Matrix.

Unter der Kehrmatrix (inversen Matrix) einer Matrix verstehen wir die Matrix, die, mit der Originalmatrix multipliziert, die Einheitsmatrix ergibt. Betrachten wir dazu ein Beispiel:

Es sei $C = \begin{bmatrix} 4 & -3 \\ 6 & 8 \end{bmatrix}$ und $C = \begin{bmatrix} 0,16 & 0,06 \\ -0,12 & 0,08 \end{bmatrix}$

Das Produkt C · H ergibt

$$C \cdot H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

also die Einheitsmatrix. Deshalb ist C die Kehrmatrix von H und andererseits H die Kehrmatrix von C. Man schreibt

$$H = C^{-1},$$

um auszudrücken, daß H die Kehrmatrix von C ist. Gehen wir zu unserem Gleichungssystem zurück, so können wir die Matrizengleichung $C \cdot X = K$ auf beiden Seiten mit C^{-1} multiplizieren. Auf diese Weise erhalten wir:

$C^{-1}CX = C^{-1}K$, und da $C^{-1}C = E$ ist, folgt daraus:
$X = C^{-1}K$

Setzen wir die Werte ein, so ergibt sich

$$X = \begin{bmatrix} 0,16 & 0,06 \\ -0,12 & 0,08 \end{bmatrix} \cdot \begin{bmatrix} 11 \\ 9 \end{bmatrix}$$

Führen wir die Multiplikation durch, so bekommen wir für X folgenden Lösungsvektor

$$X = \begin{bmatrix} 2,30 \\ -0,60 \end{bmatrix}$$

Damit ist das gegebene Gleichungssystem gelöst. Wir haben somit eine Methode entwickelt, nach der wir Systeme linearer Gleichungen lösen können, indem wir die Matrix der Koeffizienten aufstellen und dann aus ihr die Kehrmatrix entwickeln. In der Praxis ist dieser Weg meist nicht der eleganteste oder leichteste Weg zur Lösung linearer Gleichungssysteme.

Es gibt verschiedene Möglichkeiten, die inverse Matrix zu einer gegebenen Matrix zu bilden, doch würde eine Behandlung dieser Methoden den Rahmen dieses Buches sprengen.

Aufgabengruppe 4

1) Berechne für die folgenden Zahlen den Mittelwert, die Varianz, die Standardabweichung und den mittleren Fehler: 56, 59, 59, 63, 63.

2) Für die folgenden Daten sind $\bar{X}$, s^2, s und M zu berechnen:

x	f
870 bis 876	3
877 bis 883	12
884 bis 890	14
891 bis 897	14
898 bis 904	7

3) Für die folgenden Daten sind r^2 und r zu berechnen sowie a und b zu bestimmen (durch a und b ist die gerade Linie y=a+bx gegeben):

x	83	86	84	82	80
y	55	58	62	64	66

4) Gegeben sind die Matrizen:

$$R = \begin{bmatrix} -2 & 4 & 6 \\ 1 & -3 & 5 \end{bmatrix}, \quad S = \begin{bmatrix} 1 & 2 \\ -3 & -5 \end{bmatrix} \text{ und } T = \begin{bmatrix} 6 & -2 \\ -4 & 7 \end{bmatrix}$$

Gesucht sind folgende Matrizen:

a) $-3 \cdot R$ b) $S - T$ c) $S + 3T$ d) $S \cdot R$ e) R'

5) Unter Benutzung der Matrizen von Aufgabe 4) ist die folgende Problemstellung zu lösen: Wenn wir die Gleichungssysteme

$$SX_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \text{und} \quad SX_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

auflösen, so erhalten wir als Lösungen

$$X_1 = \begin{bmatrix} -5 \\ 3 \end{bmatrix} \quad \text{und} \quad X_2 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$

(Unter X ist die Matrix $\begin{bmatrix} x \\ y \end{bmatrix}$ zu verstehen). Bilden wir nun eine Matrix U zu

$$U = \begin{bmatrix} -5 & -2 \\ 3 & 1 \end{bmatrix},$$

so erhalten wir bei der Multiplikation von S mit U die Einheitsmatrix E. Durch Benutzung dieser Methode soll nun versucht werden, auch die Kehrmatrix von T zu entwickeln.

6) Unter Benutzung der bei Aufgabe 4) gegebenen Matrizen sollen die Matrixprodukte $S \cdot T$ und $T \cdot S$ gebildet werden. Es ist festzustellen, ob die beiden Produkte gleich sind!

2.10. Logarithmen

Schreiben wir die Beziehung $N=b^x$ nieder, so unterstellen wir damit, daß die Zahl b, Basis genannt, zur x-ten Potenz erhoben werden soll; x heißt Exponent. Betrachten wir den Exponenten x als unbekannt, so ist es meist zweckmäßiger, diese Beziehung in einer anderen Form, nämlich aufgelöst nach x, niederzuschreiben. Wir schreiben

$$x = {}^b\log N$$

und lesen es zu „x ist gleich der Logarithmus von N zur Basis b". Beide Formen der Gleichung haben die gleiche Bedeutung; die gleiche Beziehung zwischen den drei Größen wird nur durch eine unterschiedliche Schreibweise ausgedrückt.

Ist x eine ganze Zahl, so können wir b^x durch wiederholte Multiplikation von b berechnen. Z. B. ist $10^3=10 \cdot 10 \cdot 10=1000$ und $3^4=3 \cdot 3 \cdot 3 \cdot 3=81$. Ist x keine ganze Zahl, so können wir N nicht auf eine solche einfache Weise berechnen. Wir können freilich meist sofort erkennen, daß $4^{0,5}=2$ ist und mit Hilfe einer Tafel der Kubikwurzeln auch, daß $7^{1/3}=1,9129$ ist; im allgemeinen wird jedoch die Berechnung von Potenzen, deren Exponent nicht ganzzahlig ist, mit Hilfe der Logarithmen vorgenommen.

Für das Rechnen mit Logarithmen gelten die nachstehenden vier Logarithmengesetze:

1) ${}^b\log(m \cdot n) = {}^b\log m + {}^b\log n$
2) ${}^b\log(m : n) = {}^b\log m - {}^b\log n$
3) ${}^b\log(M^x) = x \cdot {}^b\log M$
4) ${}^b\log(M) = {}^c\log M \cdot {}^b\log c$

Hierbei sind b bzw. c irgendwelche Basen, die jedoch positiv und nicht gleich 1 sein müssen.

Logarithmentafeln für Logarithmen zur Basis 10 (dekadische oder gewöhnliche Logarithmen) sind überall verfügbar [1]. Unter Verwendung dieser Tafeln können wir Ausdrücke wie $7^{0,25}$ leicht ermitteln. Zunächst bestimmen wir den Logarithmus von $7^{0,25}$, indem wir schreiben:

$$\lg(7^{0,25}) = 0,25 \cdot \lg(7) = 0,25 \cdot 0,84510 = 0,21128$$

1) Auch als Briggsche Logarithmen bezeichnet.

(Schreibt man, wie wir es getan haben, nur lg, so versteht man darunter Logarithmen zur Basis 10). Wir suchen dann in der Logarithmentafel diejenige Zahl auf, deren Logarithmus gleich $0,21128$ ist und finden $\lg(1,6265)=0,21128$. Somit ergibt sich, daß $7^{0,25}=1,6265$ ist. Verwenden wir zur Berechnung Logarithmentafeln mit einer größeren Stellenzahl, so können wir Resultate erhalten, die bis auf die gewünschte Stellenzahl genau sind. Schlägt man in der Tafel nach, welcher Wert sich für N ergibt, wenn die Beziehung $\lg(N)=0,21128$ vorliegt, so bezeichnet man dieses Vorgehen als das Aufsuchen des Numerus von $0,21128$. Wir können diesen Sachverhalt auch wie folgt darstellen:

Da $b^x=N$ ist, können wir dafür auch

$$b^x = b^{{}^b\log(N)}$$

schreiben. Somit bedeutet letzten Endes das Aufsuchen des Numerus die Beantwortung der Frage, welcher Exponent der Basis b gegeben werden muß; der Exponent ist in diesem Fall gleich dem Logarithmus von N.

Obwohl jede positive Zahl mit Ausnahme von 1 als Basis verwendet werden kann, wird die Basis 10 deshalb am meisten benutzt, weil sie die Verwendung von Logarithmentafeln wesentlich vereinfacht. Technische und wissenschaftliche Anwendungen erfordern jedoch oft die Berechnung von Formeln, die beträchtlich einfacher gehandhabt werden können, wenn man an Stelle des gewöhnlichen Logarithmus den natürlichen Logarithmus einsetzt. Die Basis dieser Logarithmen ist e, ein unendlicher nichtperiodischer Dezimalbruch (wie π). Der Zahlenwert von e beträgt $2,7182818284\ldots$. Für die natürlichen Logarithmen ist die Schreibweise ln genormt, also z.B. $\ln 7$. Zwischen den natürlichen und den dekadischen Logarithmen besteht folgende Beziehung:

$$\lg(N) = 2,3026 \cdot \ln(N)$$

Diese Beziehung kommt dadurch zustande, daß $\ln 10 = 2,3026$ ist (auf vier Stellen genau). Die Klammern um N können selbstverständlich auch weggelassen werden.

Von der Zahl 0 kann kein Logarithmus gebildet werden, da keine Potenz einer positiven Zahl den Potenzwert 0 ergibt. Ebenso können keine Logarithmen negativer Zahlen gebildet werden. Verwenden wir also in Berechnungen Logarithmen, so müssen wir sorgfältig darauf achten, daß wir keinen Versuch unternehmen, die Logarithmen von negativen Zahlen oder von 0 zu bilden.

2.11. Trigonometrie

Wir wollen zunächst ein rechtwinkliges kartesisches Koordinatensystem einen Winkel einzeichnen, dessen Scheitelpunkt im Koordinatenursprung (Schnittpunkt der beiden Achsen) liegt und dessen einer Schenkel mit der positiven x-Achse zusammenfällt. Auf dem anderen Schenkel wählen wir uns einen beliebigen Punkt mit dem Koordinaten (x;y) aus. Ein solcher Winkel ist in Abb. 2.2. dargestellt.

Die Entfernung des Punktes (x;y) vom Ursprung ergibt sich nach dem Satz von Pythagoras zu $r=\sqrt{x^2+y^2}$. Bezeichnen wir den Winkel mit α, so können sechs verschiedene trigonometrische Funktionen durch die nachstehenden Bezeichnungen definiert werden:

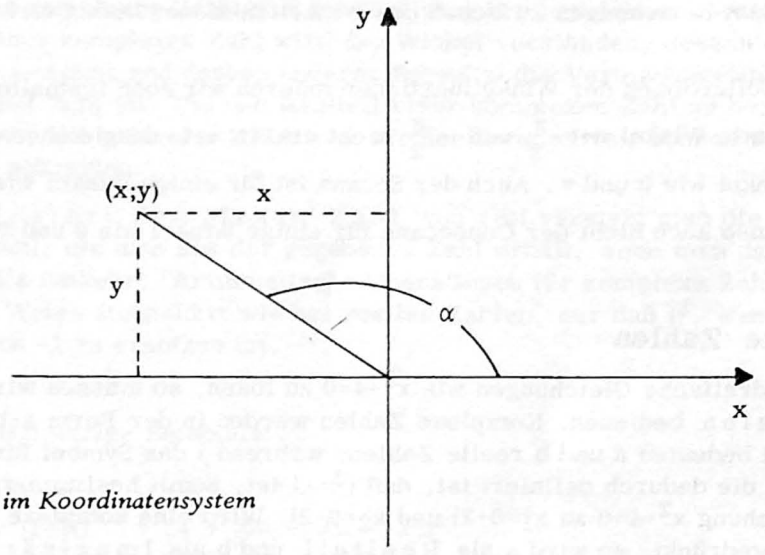


Abb. 2.2. Winkel im Koordinatensystem

Sinus von α:	$\sin \alpha = \frac{y}{r}$
Cosinus von α:	$\cos \alpha = \frac{x}{r}$
Tangens von α:	$\tan \alpha = \frac{y}{x}$
Cotangens von α:	$\cot \alpha = \frac{x}{y}$
Secans von α:	$\sec \alpha = \frac{r}{x}$
Cosecans von α:	$\operatorname{cosec} \alpha = \frac{r}{y}$

Zwischen diesen Funktionen können verschiedene Beziehungen und Identitäten hergeleitet werden. So gilt z. B. für jeden beliebigen Winkel α, daß

$$\sin^2\alpha + \cos^2\alpha = 1 \text{ ist.}$$

(Unter $\sin^2\alpha$ versteht man $(\sin\alpha)^2$; diese Schreibform ist in der Mathematik eingeführt worden, um unnötige Klammern zu vermeiden). Ebenso gilt beispielsweise

$$1 + \tan^2\alpha = \sec^2\alpha \quad ,$$

$$\tan\alpha = \frac{\sin\alpha}{\cos\alpha} \qquad \text{und}$$

$$\sin(180^\circ + \alpha) = -\sin\alpha \quad .$$

Winkel können in Grad gemessen werden, aber sicherlich ist es oft zweckmäßiger, Winkel in Radianten, also im Bogenmaß, anzugeben. Zur Umrechnung vom Gradmaß aufs Bogenmaß bedient man sich der Beziehung $180^\circ \mathrel{\hat{=}} \pi$ rad. Somit entspricht $1 \text{ rad} \mathrel{\hat{=}} \frac{180^\circ}{\pi}$ und $1^\circ \mathrel{\hat{=}} \frac{\pi}{180}$ rad $= 0,01745$ rad. In diesem Buch wollen wir alle Winkel in rad angeben, sofern wir nicht ausdrücklich vermerken, daß das Gradmaß zu nehmen ist.

Mit den verschiedenen Beziehungen zwischen den Winkelfunktionen befaßt sich die Trigonometrie.

Ergänzend zu den Definitionen der Winkelfunktionen müssen wir noch festhalten, daß der Tangens für bestimmte Winkel wie $\frac{\pi}{2}$ und $-\frac{\pi}{2}$ nicht erklärt ist, desgleichen der Cotangens nicht für solche Winkel wie 0 und π. Auch der Secans ist für einige Winkel wie $\frac{\pi}{2}$ und $-\frac{\pi}{2}$ nicht definiert, ebenso auch nicht der Consecans für einige Winkel wie 0 und π.

2.12. Komplexe Zahlen

Versuchen wir quadratische Gleichungen wie $x^2+4=0$ zu lösen, so müssen wir uns komplexer Zahlen bedienen. Komplexe Zahlen werden in der Form a+bi geschrieben. Hierbei bedeuten a und b reelle Zahlen, während i das Symbol für die imaginäre Einheit ist, die dadurch definiert ist, daß $i^2=-1$ ist. Somit bestimmen sich die Lösungen der Gleichung $x^2+4=0$ zu $x_1=0+2i$ und $x_2=0-2i$. Wird eine komplexe Zahl in der Form a+bi ausgedrückt, so wird a als Realteil und b als Imaginärteil der komplexen Zahl bezeichnet. Ist a=0, jedoch b≠0, so sagen wir, daß wir es mit einer reinen Imaginärzahl zu tun haben. Ist umgekehrt a≠0 und b=0, so erhalten wir die reellen Zahlen als Teilmenge der komplexen Zahlen.

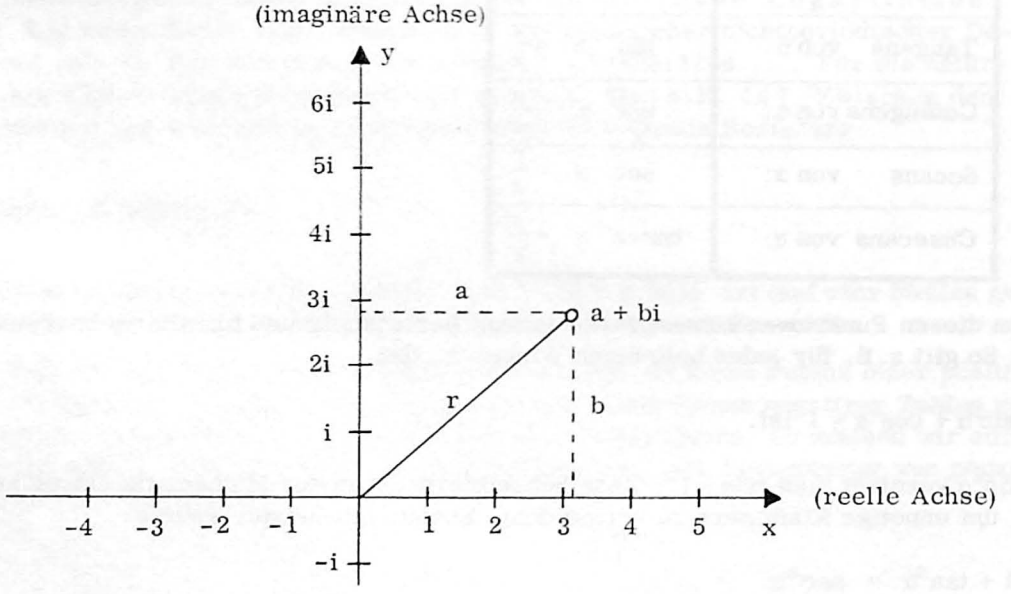


Abb. 2.3. Zeichnerische Darstellung einer komplexen Zahl

Im Gegensatz zu den reellen Zahlen ist es uns nicht möglich, die komplexen Zahlen in einer bestimmten Reihenfolge anzuordnen. Daher können wir Vergleichssymbole wie <, = usw. nicht verwenden, wenn wir uns mit komplexen Zahlen befassen [1]. Der Absolutbetrag oder Modul einer komplexen Zahl a+bi ist definiert zu $\sqrt{a^2+b^2}$, d.h. als die Entfernung des Punktes (a;b) vom Ursprung. Die Abb. 2.3. zeigt eine in der Gaußschen Zahlenebene dargestellte komplexe Zahl. Der Realteil a der komplexen Zahl wird auf der x-Achse, der Imaginärteil b auf der y-Achse aufgetragen.

1) Komplexe Zahlen können nur einander gleich oder ungleich sein.

Der Modul einer komplexen Zahl wird gewöhnlich mit r bezeichnet. Unter dem Argument einer komplexen Zahl wird der Winkel verstanden, dessen einer Schenkel die positive x-Achse und dessen anderer Schenkel die Verbindungslinie vom Ursprung zum Punkt (a;b) ist. Um den Realteil einer komplexen Zahl zu bezeichnen, verwenden wir die Schreibweise R(a+bi)=a; für den Imaginärteil können wir entsprechend I(a+bi)=b schreiben.

Unter der konjugiert komplexen Zahl von a+bi versteht man die Zahl a-bi, d.h. also die Zahl, die man aus der gegebenen Zahl erhält, wenn man das Vorzeichen des Imaginärteils umkehrt. Arithmetische Operationen für komplexe Zahlen werden in der gleichen Weise ausgeführt wie bei reellen Zahlen, nur daß i^2, wenn es auftritt, stets durch -1 zu ersetzen ist.

Man betrachte dazu einige Beispiele:

a) $(3+4i) + (8-7i) = 11-3i$

b) $(3+4i)\cdot(8-7i) = 24 + 32i - 21i - 28i^2$
$= 24 + 11i + 28$
$= 52+11i$

c) $(3+4i):(8-7i) = (3+4i)(8+7i) : (8-7i)(8+7i)$
$= (-4+53i):113$
$= -\frac{4}{113} + \frac{53}{113}\,i$

Das letzte Beispiel zeigt, wie man bei der Division zweier komplexer Zahlen vorzugehen hat (Erweiterung des Bruches mit der konjugiert komplexen Zahl des Nenners).

Aufgabengruppe 5

1) Im Dreieck ABC ist der bei C liegende Winkel ein rechter Winkel; die A gegenüber liegende Seite ist 9 cm und die B gegenüberliegende Seite 40 cm lang. Zeichne dieses Dreieck in ein rechtwinklig kartesisches Koordinatensystem so ein, daß der Eckpunkt C im Ursprung liegt und der eine Schenkel des rechten Winkels mit der positiven x-Achse zusammenfällt. Bestimme dann folgende Werte der Winkelfunktionen:

   a) sin (CAB) b) sec (CAB) c) cos (CAB) d) sin (ABC) e) cot (ABC) f) sin (BCA)

2) Berechne folgende Werte:

   a) $^4\log 16$ b) $^8\log 32$ c) lg 0,01 d) ln 100

3) Bestimme x aus den Beziehungen

   a) $^{16}\log x = 2$ b) $^x\log 32 = 2,5$ c) $^5\log x = 0$ d) $^4\log x = -2$

4) Bestimme die Resultate der nachfolgenden Rechenoperationen mit komplexen Zahlen:

   a) $(3-8i) - (7+4i)$ b) $(3-8i)\cdot(7-4i)$ c) $(3-8i):(7-4i)$ d) $(1+2i)^3$

3. Der Aufbau und die Wirkungsweise von Computern

3.1. Programme und Programmabläufe

Wir wollen jetzt einmal unsern Blick auf einen Autohändler werfen, der mehrere Verkäufer beschäftigt, die sich mit dem Vertrieb der beiden Wagentypen „Ritterburg" und „Raubvogel" befassen. Jeder Verkäufer erhält ein festes Monatsgehalt von 1800.-DM; dazu kommt für jedes verkaufte Auto eine Provision in Höhe von 120.-DM für „Ritterburg"-Wagen und von 160.-DM für „Raubvogel"-Wagen. Jeder Verkäufer muß am Ende eines jeden Monats seinen monatlichen Verkaufsbericht vorlegen, der mit der Aufstellung der Lageränderungen verglichen wird. Erst danach wird dieser als Basis zur Berechnung seiner Provisionen verwendet. Da der Händler im allgemeinen zu viel zu tun hat, um die Provisionen selbst berechnen zu können, hat er damit einen Büroangestellten beauftragt. Leider kann er nicht dafür garantieren, daß jeden Monat der gleiche Angestellte diese Arbeit verrichtet. Aus diesem Grunde hat er eine genaue Arbeitsanleitung verfaßt, deren einzelne Anweisungen die Arbeit so exakt beschreiben, daß es keine Rolle mehr spielt, wer mit der Monatsabrechnung betraut wird. Jeder Angestellte ist fortan in der Lage, sie korrekt auszuführen. Eine solche Arbeitsanleitung könnte z.B. folgende Instruktionen umfassen:

1) Greife den monatlichen Verkaufsbericht irgendeines Verkäufers heraus. Lies die Anzahl der verkauften „Ritterburg"-Wagen aus der Spalte A des Formulars ab und die Anzahl der verkauften „Raubvogel"-Wagen aus Spalte B.
2) Berechne die sich aus Spalte A ergebende Provision durch Multiplikation der in A stehenden Zahl mit 120.
3) Berechne die sich aus Spalte B ergebende Provision durch Multiplikation der in B stehenden Zahl mit 160.
4) Berechne den gesamten fälligen Verdienst durch Bildung der Summe aus 1800 und den unter 2) und 3) berechneten Provisionen.
5) Trage den Gesamtverdienst neben den Namen des Verkäufers in die Gehaltsliste ein.
6) Wiederhole den Vorgang ab Schritt 1) mit dem monatlichen Verkaufsbericht des nächsten Verkäufers.

Eine Menge solcher Instruktionen heißt ein P r o g r a m m zur Berechnung des Monatsverdienstes von Verkäufern. Im allgemeinen versteht man unter einem Programm eine vollständige Menge von Instruktionen, die zur Lösung eines bestimmten Problems dienen. Es sollte alle Möglichkeiten des Problems berücksichtigen und nichts als selbstverständlich annehmen. Mitunter ist es weitaus zweckmäßiger, ein Programm zu veranschaulichen, anstatt daß wir die Instruktionen in Worte kleiden. Für unser Problem haben wir eine solche Veranschaulichung in Abb. 3.1. vorgenommen. Eine Darstellung dieser Art heißt P r o g r a m m a b l a u f p l a n, abgekürzt PAP.

Man wird sofort bemerken, daß wir in unserem Programmablaufplan eine zusätzliche Instruktion gegenüber der Arbeitsanleitung aufgenommen haben, indem wir dem Büroangestellten auch sagen, was er zu tun hat, wenn er alle Verkaufsberichte aufgearbeitet hat. Er soll nämlich seine Arbeit beenden. Haben wir nun wirklich schon alle Möglichkeiten berücksichtigt? Vielleicht wird der Büroangestellte beim Lesen eines Verkaufsberichtes feststellen, daß der Verkäufer vergessen hat, seinen Namen oder eine andere wichtige Information in den Verkaufsbericht einzutragen. Fehlen für diese Sonderfälle Instruktio-

nen, so geht der Händler damit ein gewisses Risiko ein, denn der Büroangestellte könnte dann zu Arbeitsschritten veranlaßt werden, die er nicht tun sollte, wie z.B. Wegwerfen des Berichtes in den Papierkorb oder Ausfüllen der lückenhaften Teile des Berichtes nach eigenem Gutdünken. Deshalb sollte es als wünschenswert erachtet werden, zusätzliche Instruktionen zwischen der ersten und der zweiten Instruktion ins Programm einzufügen, die dem Büroangestellten veranlassen, das Formular auf Vollständigkeit der Ausfüllung zu überprüfen und es dem Händler zu übergeben, falls es fehlerhaft oder lückenhaft ausgefüllt sein sollte.

In vieler Hinsicht arbeitet der Büroangestellte wie eine Maschine: Wir geben ihm Zahlenpaare (Verkaufszahlen in Spalte A und in Spalte B) ein, und er gibt uns nach einer kurzen, durch seine eigenen Arbeitsverrichtungen bedingten Pause Werte zurück, nämlich die Gesamtverdienste der Verkäufer. Jede Maschine, die eine derart gestaltete Arbeit verrichtet, wird als Computer bezeichnet. Im Grunde genommen, können alle Operationen, die ein Computer durchführen kann, genau so wie die Arbeitsschritte des Büroangestellten in vier verschiedene Gruppen eingeteilt werden:

1) Eingabe/Ausgabe, kurz EA
   Der Büroangestellte liest Zahlen von einem Verkaufsbericht ab und schreibt Zahlen auf eine Gehaltsliste.

2) Speichern oder ins Gedächtnis aufnehmen
   Nachdem der Büroangestellte im 2. Arbeitsschritt die Provision für A berechnet hat, muß er sich das Resultat in seinem Gedächtnis solange merken (oder, wenn er das vorzieht, auf einen Schmierzettel notieren), bis er es im Schritt 4) wieder benötigt. Er muß sich ebenfalls immer an die Instruktionen erinnern, denen er bei seiner Arbeit zu folgen hat. Er wird deshalb sinnvollerweise immer eine Abschrift von ihnen zur Verfügung halten.

3) Arithmetische Operationen
   Der Büroangestellte hat hier Multiplikationen und Additionen zu erledigen.

4) Steuerungsoperationen
   Darunter versteht man bei der manuellen Arbeit die Instruktionen, die dem Büroangestellten sagen, was er als nächsten Schritt zu tun hat. In unserem Fall wird ihm durch die 6. Instruktion mitgeteilt, daß er mit dem Lesen des nächsten Verkaufsberichtes fortfahren soll, falls er noch nicht alle bearbeitet hat.

Ein elektronischer Computer ist nichts mehr und nichts weniger als ein elektronischer Assistent, der in der Lage ist, die eben erwähnten vier Funktionen auszuführen. Die Eingabe/Ausgabe kann in ihrer einfachsten Form aus Stromstößen bestehen, die von einer elektrischen Schreibmaschine aufgenommen werden bzw. die

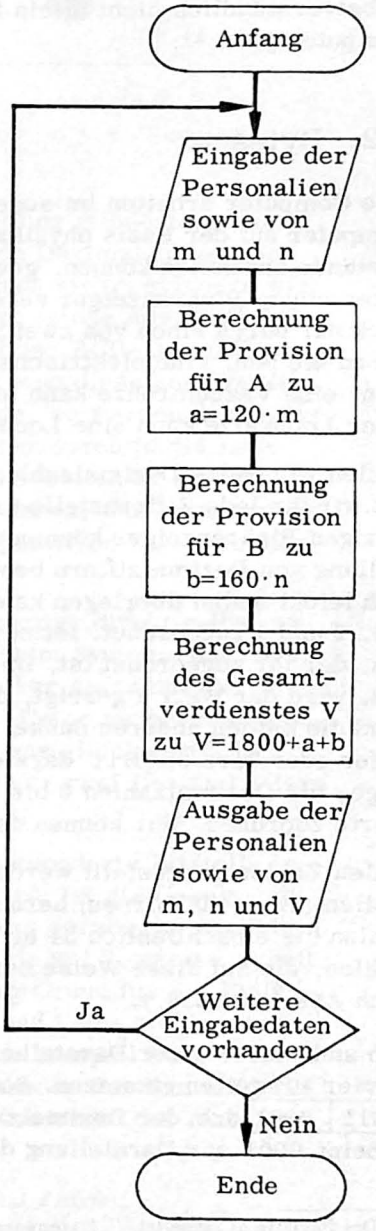


Abb. 3.1. Programmablaufplan

für eine elektrische Schreibmaschine bestimmt sind. Zur Speicherung können magnetisierte Ferritkerne benutzt werden, die die Ziffern unserer Zahlen repräsentieren. Die arithmetische Einheit könnte ein elektronischer Tischrechner sein und das Steuerwerk schließlich könnte dem Kanalwähler eines Fernsehgerätes gleichen, der automatisch auf die nächste Instruktion eingestellt wird, außer wenn ein Vorgang ähnlich dem Schritt 6 unseres Programmes ablaufen soll, bei dem es erforderlich wäre, den Kanalwähler auf eine neue Anfangsposition einzustellen.

Die technische Realisierung der soeben aufgezeigten logischen Funktionen ist bei den einzelnen Computertypen recht unterschiedlich. Unsere Diskussionen sollen auf dem Datenverarbeitungssystem IBM /370 fußen, einem System, das weltweit mit am häufigsten installiert und eingesetzt ist. Die grundlegenden Gedankengänge jedoch gelten selbstverständlich nicht allein für das System IBM /370, sondern auch für alle anderen Computertypen [1].

3.2. Bytes

Alle Computer arbeiten im sogenannten binären Modus. Darunter verstehen wir, daß die Computer auf der Basis physikalischer Größen operieren, die nur zwei ausgezeichnete Zustände annehmen können, gewöhnlich als Zustand 0 und als Zustand 1 bezeichnet. Unter einem Binäranzeiger versteht man jede Einrichtung, die in einem bestimmten Augenblick nur durch einen von zwei möglichen Zuständen ausgezeichnet ist. Um bei Beispielen zu bleiben, eine elektrische Glühbirne kann entweder ausgeschaltet oder eingeschaltet sein, eine Vakuumröhre kann leitend oder nichtleitend sein, auf einer bestimmten Stelle einer Lochkarte kann eine Lochung vorliegen oder es kann keine vorhanden sein.

Wollen wir ganze Dezimalzahlen durch Binäranzeiger darstellen, so ist es erforderlich, daß wir für jede Ziffernstelle mehr als einen Binäranzeiger vorsehen, denn durch einen einzigen Binäranzeiger können nur zwei Ziffern dargestellt werden. Zu einer solchen Darstellung von Dezimalziffern benötigen wir nicht weniger als vier Binäranzeiger, wie man sich leicht selbst überlegen kann. In Abb. 3.2. haben wir den vier Glühbirnen die Werte 8, 4, 2 und 1 zugeordnet. Ist eine Birne eingeschaltet, so soll sie den Wert repräsentieren, der ihr zugeordnet ist, im anderen Fall soll sie als 0 gezählt werden. In der Abb. 3.2. wird der Wert 6 gezeigt, da die Glühbirnen mit den Werten 4 und 2 leuchten, während die beiden anderen dunkel sind. Man sieht also, daß jede Glühbirne eine Dualziffer oder kurz ein Bit darstellen kann [2]. Mit vier Birnen sind wir also in der Lage, die Dezimalzahlen 0 bis 15 auszudrücken, wenn wir den einzelnen Birnen feste Werte zuordnen. Wir können deshalb 6 auch zu 0110 schreiben (siehe Abb. 3.2.).

Sollen Zahlen dargestellt werden, die größer als 15 sind, so müssen wir weitere Bitstellen (hier: Glühbirnen) heranziehen. Mit fünf Bitstellen lassen sich alle Dezimalzahlen bis einschließlich 31 erfassen, denn $11111_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 31_{10}$. Zahlen, die auf diese Weise dargestellt werden, heißen bekanntlich Dualzahlen (siehe auch Abschnitt 2.3.).

Ein anderer Weg zur Darstellung von Zahlen besteht darin, daß wir für jede Dezimalstelle vier Bitstellen einsetzen. Somit würde die Dezimalzahl 31 dargestellt werden durch | 0011 | 0001 | d. h. der Dezimalziffer 3 würde 0011 entsprechen, daran anschließend erscheint 0001 zur Darstellung der Dezimalziffer 1. Diese Darstellung ist als BCD-Dar-

1) Die Begriffe „Computer", „Datenverarbeitungsanlage", „Datenverarbeitungssystem" und „Elektronische Rechenanlage" bedeuten gemeinhin das gleiche.

2) Bit ist die Abkürzung von binary digit (Binärziffer, besser Dualziffer im deutschen Sprachraum).

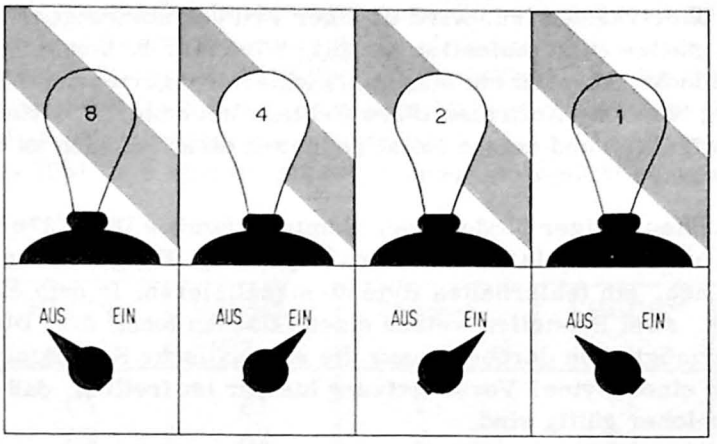


Abb. 3.2. Darstellung der Dezimalziffer 6

stellung [1] bekannt. Es ist zu beachten, daß die BCD-Darstellung mindestens genau so viele Bitstellen erfordert wie die reine Dualdarstellung von Dezimalzahlen nach ihrer Konvertierung ins Dualsystem. Gewöhnlich sind es jedoch mehr; erinnern wir uns, daß jeder Dezimalstelle etwa 3,3 Bitstellen entsprechen (siehe Abschnitt 2.3.), wenn Dezimalzahlen ins Dualsystem konvertiert werden. Bei der BCD-Darstellung sind es jedoch, wie wir soeben gesehen haben, stets 4. Baut man diese zusätzlich erforderlichen Bitstellen in eine Datenverarbeitungsanlage ein, so werden deren Herstellungskosten ansteigen, doch vergessen wir nicht, daß wir dadurch in die Lage versetzt werden, Computer auf der Basis von Dezimalzahlen arbeiten zu lassen und nicht auf der Basis von Dualzahlen. Wir werden in Kapitel 4 sehen, daß beim System IBM /370 beide Darstellungsarten vorliegen, so daß wir die positiven Eigenschaften beider Zahlensysteme sinnvoll ausnutzen können.

Die kleinste Informationseinheit, die normalerweise der Computer direkt selbst ansprechen kann, heißt Byte [2]. Ein solches Byte setzt sich beim System IBM /370 aus neun Bitstellen zusammen. Acht dieser Bitstellen dienen zur Aufnahme der Information, die neunte Bitstelle zur Aufnahme eines Prüfbits. Unter anderem können diese acht Bitstellen acht Dualziffern repräsentieren, oder, was gleichwertig ist, zwei Sedezimalziffern oder sie können auch zur Darstellung von zwei Dezimalziffern in BCD-Form herangezogen werden.

Die Bitstelle, die zur Aufnahme des Prüfbits dient, ist eine gesonderte Bitstelle innerhalb eines Bytes, die wie folgt auf 1 oder auf 0 gesetzt wird: Ist die Gesamtzahl der dualen Einsen[3] auf den anderen acht Bitstellen eines Bytes gerade, so wird die Prüfbitstelle auf 1 gesetzt, ist sie jedoch ungerade, so wird sie auf 0 gesetzt. Somit ist die Anzahl der Einsen in einem Byte immer ungerade. Der Grund für die Einführung dieses Prüfbits ist selbstverständlich darin zu suchen, daß keine mechanische oder elektronische Einrichtung völlig betriebssicher ist. Zum Beispiel kann eine Glühbirne ausbrennen. Die IBM /370 verwendet eine Prüfmethode, die man als Prüfung auf ungerade Parität bezeichnet; immer, wenn ein Byte innerhalb der Maschine

1) BCD ist die Abkürzung von binary coded decimal (binärverschlüsselte Dezimal-Ziffern).
2) Ein Byte belegt gerade eine Speicherstelle.
3) Gewöhnlich pflegt man schlechthin von einem Bit dann zu sprechen, wenn eine duale 1 repräsentiert vorliegt; eine duale 0 wird einfach als 0 bezeichnet.

angesprochen oder übertragen wird, wird es einer Prüfung unterzogen, um festzustellen, ob es eine gültige Bitkombination enthält. Wird eine Bitkombination mit gerader Parität entdeckt, so wird ein Maschinenfehleranzeiger eingestellt, der die Maschinenbedienung über das Auftreten eines Fehlers informiert. Solche Fehler sind jedoch höchst ungewöhnlich und zeigen im allgemeinen einen Schaden an der Maschine an.

Bei den Speichergeräten einiger Modelle der Computerfamilie IBM /370 werden andere Prüfmethoden verwendet als die der Paritätsprüfung. Einige dieser Techniken sind sogar in der Lage, ein fehlerhaftes Byte zu signalisieren, in dem ein Zweibitfehler vorliegt, d.h. zwei Bitstellen weisen einen falschen Inhalt auf. Die meisten dieser Techniken ermöglichen darüberhinaus die automatische Korrektur jedes Einbitfehlers innerhalb eines Bytes. Voraussetzung hierfür ist freilich, daß die benachbarten Bytes im Speicher gültig sind.

3.3. Alphanumerische Informationen

Die Fähigkeit von Computern, arithmetische Operationen auszuführen, ist außerordentlich wichtig, dennoch ist es für den Nutzwert eines Computers mindestens ebenso wichtig, daß er auch alphabetische Informationen speichern und handhaben kann. Würden wir z.B. einen Computer für das in Abschnitt 3.1. entworfene Programm einsetzen, so müßte der Name des Verkäufers im Computer gespeichert werden, damit wir nach Berechnung seines Verdienstes den Computer veranlassen können, den Namen zusammen mit dem errechneten Verdienst auszudrucken.

Wir unterscheiden drei verschiedene Arten von Zeichen:

a) Numerische Zeichen oder Ziffern: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

b) Alphabetische Zeichen oder Buchstaben: A-a, B-b, C-c, D-d, E-e, F-f, G-g, H-h, I-i, J-j, K-k, L-l, M-m, N-n, O-o, P-p, Q-q, R-r, S-s, T-t, U-u. V-v, W-w, X-x, Y-y, Z-z

c) Sonderzeichen: .|,|-|+|:|;|Leerzeichen [1]|=|(|)| usw.

Unter alphanumerischen Zeichen[2] wollen wir sowohl die Buchstaben als auch die Ziffern verstehen. Wir werden ferner vielfach den Buchstaben b verwenden, um das Leerzeichen anzuzeigen, doch soll in diesem Buch jede derartige Verwendung des Buchstabens b extra vermerkt werden. Die Gesamtheit der Zeichen, die einer speziellen Maschinenausrüstung zugrunde liegt, soll als Zeichenmenge oder Zeichenvorrat[3] dieser Ausrüstung bezeichnet werden.

Da sich ein Byte aus acht Bitstellen zusammensetzt, die Prüfbitstelle nicht mitgezählt, so können wir auf einer Speicherstelle insgesamt 2^8=256 verschiedene Zeichen darstellen. Die Verschlüsselung jedes Zeichens wird am zweckmäßigsten durch die beiden Sedezimalziffern angegeben, in die das betreffende Byte zerlegt werden kann, denn zwei Sedezimalziffern belegen, wie wir gesehen haben, gerade acht Bitstellen, d.h. ein Byte. Die linke höherwertige Sedezimalziffer wird oft als Zonenteil der Zeichendarstellung bezeichnet, die rechte Ziffer dagegen als numerischer Teil. Die Verschlüsselung einiger häufig benutzter Zeichen wird in Abb. 3.3. ge-

1) Auch als Zwischenraumzeichen bezeichnet.
2) Auch als alphanumerische Zeichen bezeichnet.
3) Auch als Zeichensatz bezeichnet.

zeigt[1]. Wenn wir sagen, daß der Buchstabe I genau so verschlüsselt ist wie die beiden Sedezimalziffern C9, so meinen wir damit einfach, daß ein Byte, das den Buchstaben I enthält, das Bitmuster 1100 1001 aufweist. Aus Abb. 3.3. können wir weiterhin erkennen, daß die Zeichen 0 bis 9 zu F0 bis F9 verschlüsselt werden, d.h. daß ihr numerischer Teil gleich den Ziffern 0 bis 9 ist. Alle 256 möglichen Zeichen des Systems IBM /370 sind im Anhang A zusammengefaßt dargestellt.

Zeichen	EBCDIC	USASCII-Code	Zeichen	EBCDIC	USASCII-Code
A	C1	A1	Y	E8	B9
B	C2	A2	Z	E9	BA
C	C3	A3	0	F0	50
D	C4	A4	1	F1	51
E	C5	A5	2	F2	52
F	C6	A6	3	F3	53
G	C7	A7	4	F4	54
H	C8	A8	5	F5	55
I	C9	A9	6	F6	56
J	D1	AA	7	F7	57
K	D2	AB	8	F8	58
L	D3	AC	9	F9	59
M	D4	AD	/	61	4F
N	D5	AE	&	50	46
O	D6	AF	Leerzeichen	40	40
P	D7	B0	-	60	4D
Q	D8	B1	$	5B	44
R	D9	B2	.	4B	4E
S	E2	B3	,	6B	4C
T	E3	B4	#	7B	43
U	E4	B5	*	5C	4A
V	E5	B6	%	6C	45
W	E6	B7	ⓐ	7C	A0
X	E7	B8	'	7D	47

Abb. 3.3. Verschlüsselungen beim System IBM/370

Während einige Lochkartenmaschinen alle 256 Zeichen interpretieren können, sind die meisten Druckgeräte, wie Lochschriftübersetzer, Schreibmaschinen und Zeilendrucker auf eine Untermenge des gesamten Zeichenvorrats beschränkt, die meist zwischen 39 und 88 Zeichen umfaßt. Diese Untermengen sind stets groß genug gewählt, um alle 10 Ziffern, alle 26 Großbuchstaben und einige wichtige Sonderzeichen drucken zu können.

1) Es sind die beiden Codes EBCDIC und USASCII (siehe Abschnitt 3.4.) aufgeführt. EBCDIC ist dabei die Abkürzung von Extended Binary Coded Decimal Interchange Code (erweiterter Austausch für binär verschlüsselte Dezimalziffern).

3.4. Der Code USASCII

Der Standardcode für Informationsaustausch der USA[1] (USASCII) wurde geschaffen, um einen einheitlichen Code verwenden zu können, wenn Informationen zwischen verschiedenen Computern über Verbindungsleitungen oder andere Mittel ausgetauscht werden sollen. Dieser Code benutzt sieben Datenbitstellen und zusätzlich eine Prüfbitstelle. Dieser Code ist auch als ASCII [2] bekannt.

Das System IBM /370 kann unmittelbar mit diesem Code arbeiten. Ein Teil desselben ist ebenfalls in der Abb. 3.3. dargestellt. Da die IBM /370 in jedem Byte eine Bitstelle mehr aufweist als für die Benutzung des USASCII erforderlich ist, wird das Zweierbit des Zonenteils stets gleich dem Achterbit des Zonenteils gesetzt. Der USASCII wird mit den gleichen bereits kurz beschriebenen Prüfverfahren betrieben, u.a. ungerade Parität (siehe Abschnitt 3.2.). Der volle USASCII ist im Anhang A zusammengestellt.

Normalerweise wird jedoch die IBM /370 nicht mit diesem Code betrieben. Vielmehr ist es so, daß nach den Eingabeoperationen der USASCII in den EBCDIC [3], dem regulären Code für die IBM /370, übersetzt wird. Dadurch ist es möglich, daß alle Operationen mit dem EBCDIC ausgeführt werden. Vor den Ausgabeoperationen wird in einem solchen Fall eine Übersetzung zurück in den USASCII veranlaßt. Auf diese simple Art und Weise wird mit Daten operiert, die im USASCII vorliegen.

Wie bereits gesagt, wird der gewöhnlich benutzte (reguläre) Code beim System IBM /370 der Code sein, der technisch unter dem Begriff EBCDIC bekannt ist. Wir wollen in diesem Buch ebenfalls konsequent diesen Code benutzen.

Aufgabengruppe 6

1) Der 3-Überschußcode (Stibitzcode) verwendet zur Darstellung der Dezimalziffern fünf Bitstellen mit den Wertigkeiten C, 8, 4, 2, 1. Bevor jedoch die Ziffer verschlüsselt wird, wird zu jeder Ziffer eine 3 addiert. Somit wird die Ziffer 6 dargestellt durch 11001, d.h. zu 6+3. Stelle eine Tabelle auf, die die Verschlüsselung aller 10 Dezimalziffern enthält. Zeige, daß das Neunerkomplement jeder Ziffer gebildet werden kann, indem man auf allen Bitstellen einfach die Bits 0 und 1 austauscht (mit Ausnahme der Prüfbitstelle).

   Anmerkung: Unter C wird allgemein die Prüfbitstelle verstanden (Check···→ Prüf···).

2) Der 2-aus-5-Code verwendet zur Darstellung jeder Dezimalziffer fünf Bitstellen mit den Wertigkeiten 6, 3, 2, 1, 0. Die Bitstelle mit der Wertigkeit 0 ist eine Prüfbitstelle, die immer dann auf 1 gesetzt wird, wenn nur eine andere Bitstelle mit einer 1 belegt ist; auf diese Weise enthalten gerade immer zwei Bitstellen eine 1. Stelle eine Tabelle auf, die eine mögliche Verschlüsselung für alle Dezimalziffern enthält. Wie kann die Ziffer 0 dargestellt werden?

3) Nimm irgendeine Dualzahl her, z.B. 0110 (entspricht der 6 im Dezimalsystem), und verschiebe die Zahl um eine Stelle nach rechts, so daß die Ziffer niedrigster Ordnung verlorengeht (aus 0110 erhält man damit 011). Die beiden Zahlen, die ursprüngliche und die durch Verschiebung erhaltene, sind anschließend zu ad-

1) United States of America Standard Code for Information Interchange, abgekürzt USASCII.
2) American National Standard Code for Information Interchange, abgekürzt ASCII.
3) Extended Binary Coded Decimal Interchange Code.

dieren, wobei alle Überträge unbeachtet bleiben sollen. Man erhält dann bei unserem Beispiel:

```
0110
 011
----
0101
```

Die Verschlüsselung, die durch dieses Vorgehen erhalten wird, wird als Gray-Verschlüsselung bezeichnet. Bilde für die Zahlen von 0 bis einschließlich 15 den Gray-Code! Kann man erkennen, ob der Gray-Code einen Vorteil bietet, wenn die Gefahr besteht, daß rein zufällig ein 1-Bit gewonnen wird oder ein 1-Bit verlorengeht?

4) Lochstreifenkarten verwenden den 7-4-2-1-Code, bei dem jede Dezimalziffer durch nicht mehr als zwei 1-Bits dargestellt wird. Schreibe diese Darstellung für alle Dezimalziffern auf!

5) Welche Zeichen werden im EBCDI-Code durch die nachfolgenden Bitmuster dargestellt?

a) 1101 0001 b) 1110 0001 c) 1011 001 d) 1001 1001 e) 1010 1001 f) 1100 1001

6) Schreibe für jedes Zeichen der Aufgabe 5) das Bitmuster nieder, durch die dieses Zeichen im USASCII dargestellt wird!

3.5. Speichereinrichtungen

Glühbirnen sind keineswegs befriedigende Binäranzeiger. Sie brennen leicht aus, sind sehr materialaufwendig und erfordern umfassende elektrische Stromkreise. In Computern werden mannigfaltige Arten von Binäranzeigern verwendet; beim System IBM /370 werden zur Speicherung von Informationen drei verschiedene Arten von Binäranzeigern eingesetzt.

Besprechen wir zunächst einmal die Magnetkerne. Ein Magnetkern ist ein kleiner aus Eisenoxyd bestehender Ring von geringem Durchmesser (beim System IBM /370 beträgt der Innendurchmesser etwa 0,5 mm und der Außendurchmesser etwa 0,8 mm) [1].

Ein solcher Kern kann leicht magnetisiert werden, behält auf unbestimmte Zeit seinen magnetischen Zustand bei und kann ebenso leicht um- bzw. wieder entmagnetisiert werden. Werden die Kerne wie Perlen auf einen Draht gefädelt und wird durch den Draht ein Strom von etwa 1/2 A Stärke geschickt, so werden die Kerne magnetisiert (siehe Abb. 3.4.). Die Richtung des Magnetfeldes hängt allein von der Richtung des Stromflusses ab.

Wird der Stromfluß unterbrochen, so verbleibt der Kern in seinem magnetischen Zustand (siehe Abb. 3.5.). Nachfolgende schwache Stromstöße durch die Kerne bleiben ohne Wirkung, erst ein Strom von 1/2 A oder mehr Stärke hat zur Folge, daß entweder die ursprüngliche Magnetisierungsrichtung erhalten bleibt (wenn der Strom in der ursprünglichen Richtung fließt) oder daß das Magnetfeld des Kernes in die entgegengesetzte Richtung umklappt, was immer dann erfolgt, wenn die Stromrichtung sich geändert hat. Wir schließen uns nunmehr dem Übereinkommen an, daß Kerne, die in der einen Richtung magnetisiert sind, den binären Einsen entsprechen, Kerne, die in der entgegengesetzten Richtung magnetisiert sind, den binären Nullen.

1) Innendurchmesser 19/1000 Zoll, Außendurchmesser 31/1000 Zoll.

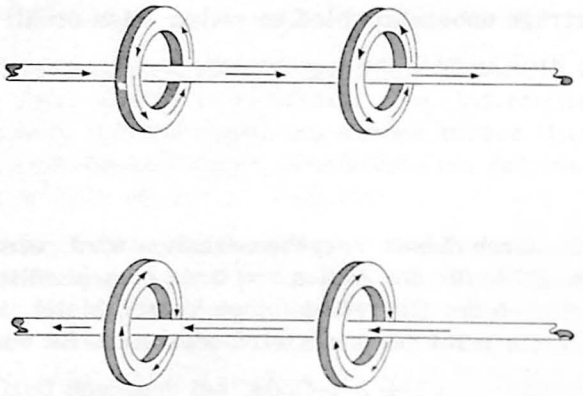


Abb. 3.4. Richtung des Magnetfeldes bei Magnetkernen

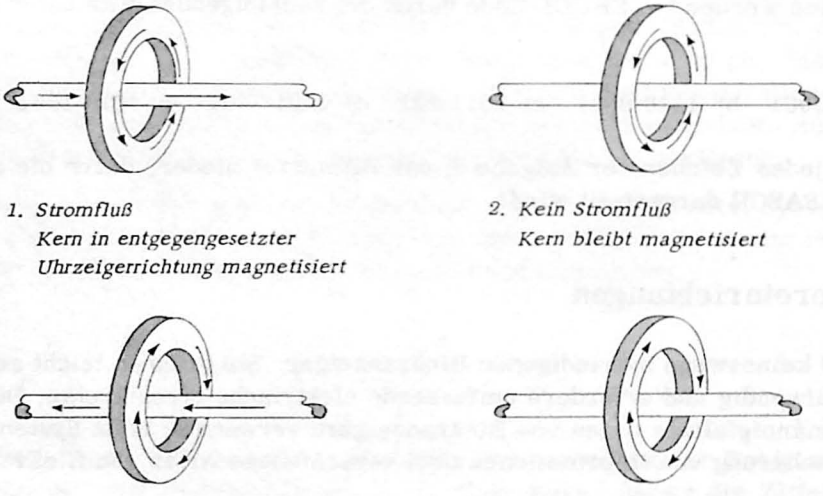


Abb. 3.5. Umkehrung der Magnetisierungsrichtung bei Magnetkernen

Ordnet man die Kerne in einer rechteckigen Matrix an, so wie es in Abb. 3.6. gezeigt wird, d.h. daß jeder Kern von zwei Drähten durchzogen ist, und schickt man durch je einen ausgewählten horizontalen und vertikalen Draht einen Stromstoß von nur 1/4 A Stärke in einer solchen Richtung, daß sich ihre magnetischen Wirkungen addieren, so wird in diesem Augenblick nur die Magnetisierungsrichtung eines einzigen Kernes beeinflußt (siehe auch Abb. 3.6.). Beim System IBM /370 enthielt eine typische Kernspeicherebene 32768 Magnetkerne.

Mit einem Binäranzeiger können wir zweierlei erledigen: Wir können ihn einmal auf einen gewünschten Zustand (0 oder 1) setzen, wir können andererseits auch nachsehen, auf welchen Zustand wir ihn zuvor gesetzt haben. Um den vorhergehenden Zustand (Status) eines Kernes zu bestimmen, schicken wir durch die Drähte, auf die er aufgefädelt ist, solche Stromstöße hindurch, die ihn in den Status 0 versetzen. Befand sich der Kern bereits im Status 0, so passiert nichts, befand er sich dagegen im Status 1, so ruft das Umklappen

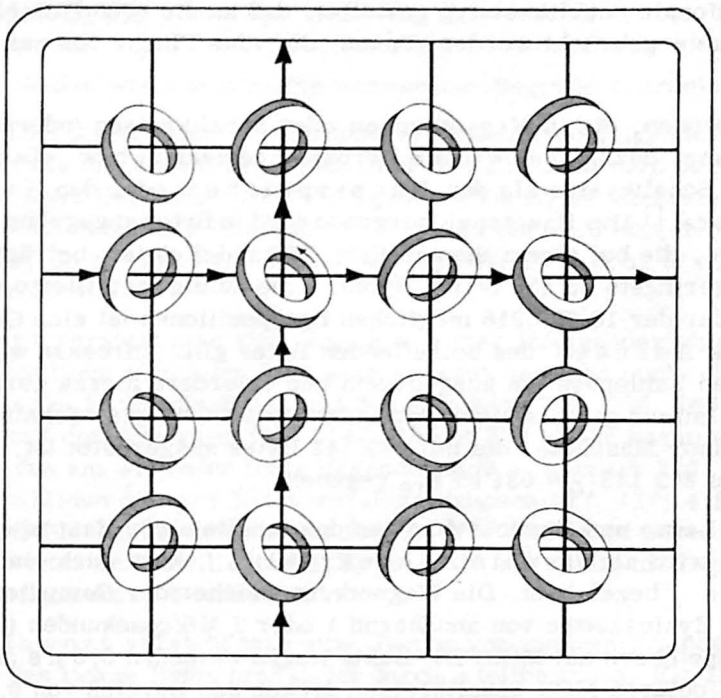


Abb. 3.6. Magnetkernebene

der Magnetisierungsrichtung des Kernes infolge der Induktionswirkung einen schwachen Stromstoß hervor, den wir in einem dritten Draht, dem sogenannten Lesedraht auffangen können. Der Lesedraht ist durch alle Kerne einer Kernspeichermatrix geführt. Da der Zustand eines Kernes zerstört wird, wenn seine Magnetisierungsrichtung und damit sein Inhalt festgestellt werden soll, so sagen wir, daß das Auslesen eines Kernes ein destruktiver Vorgang ist. Nach dem Auslesen muß selbstverständlich der Inhalt des Kernes (beim Zustand 1) durch Zurückschreiben regeneriert werden.

In einem bestimmten Augenblick kann nur ein einziger Magnetkern einer Ebene, d. h. einer Matrix, angesprochen werden. Es ist jedoch sinnvoll, gleichzeitig ein Byte oder mehrere Bytes in den Verarbeitungsprozeß einzubeziehen. Daraus resultiert die Forderung, mindestens 9 Ebenen (8 Datenbitstellen, 1 Prüfbitstelle) zu benutzen. Die meisten Modelle der Computerfamilie IBM /370 basieren auf 72 oder 144 Magnetkernebenen, so daß gleichzeitig 8 oder 16 Bytes angesprochen werden können.

Eine andere Form von Speichern, die bei Computern des Typs IBM /370 eingesetzt werden, sind Halbleiterschaltkreise aus Metalloxyden (MOS) [1]. Jeder Bitstelle entspricht hierbei ein eigener Transistorschaltkreis. Benutzt man die Technik von umfassend integrierten Schaltkreisen (LSI) [2], so können an die 2000 Bitstellen auf einem Schaltkreismodul untergebracht werden, der nur eine Fläche von 25 mm^2 bedeckt.

Der dritte Typ von Speichern, der bei der IBM /370 verwendet wird, ist der Halbleiterfeldeffekttransistor-Schaltkreis aus Metalloxyden (MOSFET) [3]. Wie bei den bipolaren MOS-Schaltkreisen entspricht auch hier jeder Bitstelle ein eigener Schaltkreis.

1) Metal - oxide - semiconductor, abgekürzt MOS.
2) Large scale integrated, abgekürzt LSI.
3) Metal - oxide - semiconductor field effect transistor, abgekürzt MOSFET.

Dieser dreidimensionale Schaltkreistyp gestattet, daß an die 8000 Bitstellen auf einem Schaltkreismodul untergebracht werden können, der eine Fläche von nur 25 mm^2 bedeckt.

Da zu den Informationen, die in Magnetkernen oder Schaltkreisen jederzeit Bezug genommen werden kann, bezeichnen wir die Kernspeichereinheit bzw. die die Bitstellen repräsentierenden Schaltkreise als den Hauptspeicher oder den Internspeicher des Computers.[1] Die Hauptspeichergröße wird in Bytes angegeben. Die höchste Hauptspeichergröße, die bei einem System IBM /370 möglich ist, beträgt 16 777 216 ($=16^6$) Bytes, die geringste 65 536 ($=16^4$) Bytes. Eine häufig installierte Größe ist 262 144 Bytes. Jeder der 16 777 216 möglichen Bytepositionen ist eine Ordnungszahl zugeordnet, die als Adresse des betreffenden Bytes gilt. Adressen werden gewöhnlich im sedezimalen Zahlensystem ausgedrückt und erfordern hierzu gerade sechs Sedezimalziffern. Immer gilt als niedrigste Adresse 000000, als nächstniedrigste 000001 usw. Bei einer Maschine, die mit 262 144 Bytes ausgerüstet ist, ist damit die höchste Adresse zu $262\,143_{10} = 03FFFF_{16}$ gegeben.

Die Zeit, die zum Lesen und Zurückschreiben des Inhalts einer Hauptspeicherposition benötigt wird, wird als Hauptspeicherzyklus , kurz auch einfach als Speicherzyklus , bezeichnet. Die Magnetkernspeicher der Computerfamilie IBM /370 besitzen Zykluszeiten von annähernd 1 oder 2 Mikrosekunden (μs). Die Zykluszeiten von Speichern auf MOSFET-Basis liegen zwischen 0,5 μs und 1,0 μs. Die Zykluszeiten von bipolaren MOS-Schaltkreisen decken den Bereich von 0,2 bis 0,9 μs ab. Spezielle Hochgeschwindigkeitsschaltkreise operieren mit Geschwindigkeiten von 54 Nanosekunden (ns)[2].

Bei bipolaren MOS-Schaltkreisen und mitunter auch bei MOSFET-Schaltkreisen ist die Zeit zum Lesen, jedoch nicht zum Ändern des Inhalts einer Speicherstelle, d. h. eines Bytes, geringfügig kürzer als der volle Speicherzyklus. Da das Lesen des Inhalts von MOS-Schaltkreisen kein destruktiver Vorgang ist, nutzen die Computerkonstrukteure den Vorteil des schnelleren Lesezyklus immer dann aus, wenn es nur irgendwie möglich ist.

Der Vorteil von Magnetkernspeichern ist darin zu sehen, daß Spannungsabfälle oder Spannungszusammenbrüche ihren Inhalt nicht zerstören können. Spannungsverluste vernichten hingegen alle in MOS-Schaltkreisen enthaltenen Informationen. MOSFET-Schaltkreise weisen die niedrigsten Herstellungskosten auf. Bipolare MOS-Schaltkreise sind die schnellsten, d. h. sie haben die kürzesten Schaltzeiten. Nachteilig sind bei ihnen jedoch die hohen Kosten, der hohe Stromverbrauch und die hohen Ansprüche hinsichtlich der Klimatisierung.

Computer des Typs IBM /370 mit Magnetkernspeicher werden nicht mehr (Stand Mitte 1973) produziert, sie sind überholt durch ähnliche Modelle mit aus monolithischen Bausteinen bestehenden Hauptspeichern auf MOSFET-Basis. Zum Schluß dieser Betrachtungen sei jedenfalls betont, daß die technische Basis des Hauptspeichers für die Programmierung im allgemeinen ohne Bedeutung ist.

Wir haben trotz der Produktionseinstellung von Magnetkernspeichern dieser Technik eine gewisse Aufmerksamkeit, ja das Schwergewicht gewidmet; begründen läßt sich das damit, weil am Beispiel der Magnetkerne am besten das Speichern von Informationen geschildert werden kann.

---

1) Auch als Arbeitsspeicher bezeichnet

2) 1 μs = 10^3 ns
1 ms = 10^3 μs
1 s = 10^3 ms

3.6. Felder und Wörter

In diesem Abschnitt wollen wir einige häufig verwendete Begriffe kennenlernen:

1) Unter einem Feld versteht man eine Gruppe von aufeinanderfolgenden Bytes. Die Feldlänge ist definiert als die Anzahl der Bytes im Feld. Ein Feld wird durch die Adresse des am weitesten links liegenden, zum Feld gehörenden Bytes adressiert. Wird z.B. auf den Speicherplätzen mit den Adressen 757 bis 760 ein Feld eingerichtet, das den Inhalt $872 besitzt, so gilt als Feldadresse die Adresse 757, d.h. des Bytes, in dem das Dollarzeichen steht.

2) Unter einem Wort versteht man eine Gruppe von vier aufeinanderfolgenden Bytes, deren am weitesten links stehendes Byte eine Adresse besitzen muß, die durch 4 teilbar ist. Zum Beispiel bilden die Bytes mit den Adressen 764, 765, 766 und 767 ein Wort. Die Bytes mit den Adressen 313, 314, 315 und 316 bilden zusammen kein Wort, weil die Adresse des am weitesten links liegenden Bytes, nämlich 313, nicht durch 4 teilbar ist; ebenso bilden die vier Bytes mit den Adressen 416, 417, 419 und 420 kein Wort, weil sie nicht lückenlos aufeinander folgen. Ein Wort wird adressiert durch die Adresse des am weitesten links liegenden Bytes, d.h. durch die niedrigste Adresse. Diese muß bekanntlich durch 4 teilbar sein.

3) Unter einem Halbwort versteht man eine Gruppe von zwei aufeinanderfolgenden Bytes; die Adresse des linken Bytes muß dabei durch 2 teilbar sein. Ein Halbwort wird adressiert, indem die Adresse des linken Bytes angegeben wird.

4) Unter einem Doppelwort versteht man eine Gruppe von acht lückenlos aufeinanderfolgenden Bytes; das am weitesten links liegende Byte, d.h. das Byte mit der niedrigsten Adresse, muß eine Adresse aufweisen, die durch 8 teilbar ist. Die Adresse eines Doppelwortes ist gleich der Adresse des am weitesten links liegenden Bytes.

In der Abb. 3.7. werden Halbwörter, Wörter, Doppelwörter, Bytes und Felder veranschaulicht.

<table>
<tr><td colspan="5">Feld mit der Adresse 800</td><td colspan="5">Feld mit der Adresse 805</td><td colspan="3">Feld mit der Adresse 810</td><td colspan="3">Feld mit der Adresse 813</td></tr>
<tr><td>Byte mit der Adresse 800</td><td>Byte 801</td><td>Byte 802</td><td>Byte 803</td><td>Byte 804</td><td>Byte 805</td><td>Byte 806</td><td>Byte 807</td><td>Byte 808</td><td>Byte 809</td><td>Byte 810</td><td>Byte 811</td><td>Byte 812</td><td>Byte 813</td><td>Byte 814</td><td>Byte 815</td></tr>
<tr><td colspan="2">Halbwort mit der Adresse 800</td><td colspan="2">Halbwort auf 802</td><td colspan="2">Halbwort auf 804</td><td colspan="2">Halbwort auf 806</td><td colspan="2">Halbwort auf 808</td><td colspan="2">Halbwort auf 810</td><td colspan="2">Halbwort auf 812</td><td colspan="2">Halbwort auf 814</td></tr>
<tr><td colspan="4">Wort mit der Adresse 800</td><td colspan="4">Wort mit der Adresse 804</td><td colspan="4">Wort mit der Adresse 808</td><td colspan="4">Wort mit der Adresse 812</td></tr>
<tr><td colspan="8">Doppelwort mit der Adresse 800</td><td colspan="8">Doppelwort mit der Adresse 808</td></tr>
</table>

Abb. 3.7. Bytes, Felder, Halbwörter, Wörter und Doppelwörter (schematische Darstellung)

Die einzelnen Bitstellen eines Bytes werden innerhalb desselben von links nach rechts fortlaufend durchnumeriert, und zwar von 0 bis 7. Um ein Beispiel zu geben, ein Byte, das den Buchstaben A (Sedezimal C1) enthält, sieht wie folgt aus (Abb. 3.8.):

	Zonenteil				Numerischer Teil			
Nummer der Bitstelle	0	1	2	3	4	5	6	7
Inhalt der Bitstelle	1	1	0	0	0	0	0	1
Inhalt der Halbbytes	C				1			
Inhalt des Bytes	A							

Abb. 3.8. Bitstellen in einem Byte

Innerhalb eines Wortes werden die Bitstellen von 0 bis 31 gezählt, innerhalb eines Halbwortes von 0 bis 15 und innerhalb eines Doppelwortes von 0 bis 63.

3.7. Seiten und Segmente

Unter einem Segment des Hauptspeichers versteht man einen zusammenhängenden Speicherblock mit 65536 oder 1048576 Speicherstellen (Bytes), der mit einer Adresse beginnt, die ein Vielfaches seines Umfanges ist. Eine Seite ist hingegen ein zusammenhängender Hauptspeicherblock, der über 2048 oder 4096 Bytes hinweg reicht und mit einer Adresse beginnt, die ein Vielfaches seines Umfanges ist.

Somit bildet bei Benutzung eines Seitenumfanges von 2048 Bytes der Hauptspeicherbereich, der bei der sedezimalen Adresse 002000 beginnt und der bei der Adresse 0027FF endet, gerade eine einzige Seite. Durch den anschließenden, bei 002800 beginnenden und bei 002FFF endenden Bereich ist die nächste Seite festgelegt. Bei einem Seitenumfang von 4096 Bytes wird durch die Speicherstelle mit der sedezimalen Adresse 002000 der Beginn einer Seite festgelegt, deren letzte Speicherstelle logischerweise dann die Adresse 002FFF besitzen muß.

Bei der Besprechung von Speichern müssen wir uns öfters mit Potenzen von 2 herumschlagen. Um die hierbei erforderlichen mühsamen Berechnungen zu umgehen, hat sich eine bestimmte Standardausdrucksweise eingebürgert, der wir uns zukünftig ebenfalls bedienen wollen. An Stelle der 10. Potenz von 2, an Stelle von 1024 also, pflegt man den Buchstaben k zu benutzen. Für die dadurch implizierte Ausdrucksweise geben wir, um sie zu untermauern, jetzt einige Beispiele:

Speicherkapazität in Bytes	Speicherkapazität in k Bytes
1 k	1024
2 k	2048
4 k	4096
64 k	65536
1024 k	1048576

Mit Ausnahme einiger weniger Probleme, über die wir in Kap. 8 sprechen werden, braucht sich der Programmierer gewöhnlich nicht mit Segmenten und Seiten zu befassen.

3.8. Register

Unter einem Register versteht man einen elektronischen Baustein, der zur vorübergehenden Speicherung von Informationen dient. In die meisten Modelle der Computerfamilie IBM /370 sind bipolare MOS-Elemente eingebaut. Die Register können nach ihrem Verwendungszweck in vier Gruppen eingeteilt werden:

1) Allgemeine Register

   Es gibt deren sechzehn, jedes kann ein Wort aufnehmen. Diese Register werden in erster Linie zur Speicherung ganzzahliger Operanden verwendet, die in Operationen der Dualarithmetik eingehen. Sie werden durchgehend von 0 bis 15 (0 bis F sedezimal) numeriert. Neben der Aufnahme von Zahlen dienen sie auch zur Aufnahme von Hauptspeicheradressen.

2) Gleitpunktregister

   Es gibt deren vier; jedes ist in der Lage, ein Doppelwort aufzunehmen. Diese Register halten die Operanden fest, die in arithmetischen Operationen in der Gleitpunktdarstellung eingehen. Sie werden wie folgt numeriert: 0, 2, 4 und 6.

3) Steuerregister

   Es gibt deren sechzehn, deren Aufnahmevermögen jeweils ein Wort beträgt. Diese Register enthalten während der Programmausführung Informationen, die die laufende Operation betreffen (Adressen, Zwischenwerte etc.). Neben diesen Informationen werden in ihnen auch die laufende Seitennummer und der Segmentumfang festgehalten. Die Steuerregister sind für den Programmierer und für den Maschinenbediener (Operateur) ohne Bedeutung. Im allgemeinen besteht auch kein Grund, sich mit ihrem Inhalt zu befassen, es sei denn, es kommt zu einem Maschinenausfall oder zum Auftreten eines gravierenden Fehlers. Für einen solchen Fall ist das Wartungspersonal so geschult, daß es die Arbeitsweise und den Verwendungszweck dieser Register zu einer sinnvollen und zeitabkürzenden Fehlersuche ausnutzen kann (selbstverständlich in Abhängigkeit von dem Modell, das es wieder funktionsfähig machen soll).

4) Interne Register

   Diese Register dienen zur Zwischenspeicherung von Informationen während der Ausführung von Instruktionen; gewissermaßen wirken sie als elektronisches Polster zwischen den anderen Funktionseinheiten der Computer. Diese Register variieren von Modell zu Modell. Für den Programmierer sind sie in keiner Weise verfügbar.

Mit Ausnahme der langsamsten Modelle der Computerfamilie IBM /370 arbeiten die Register mit einer mehrere Male höheren Geschwindigkeit als der Hauptspeicher. Da die Register im allgemeinen simultan zum Hauptspeicher arbeiten können, ist es möglich, daß zur gleichen Zeit, in der ein oder mehrere Bytes aus einem Register gelesen werden, andere Bytes vom Hauptspeicher gelesen werden. Auf diese Weise können z.B. die beiden in eine Addition eingehenden Summanden gleichzeitig herangeholt werden, vorausgesetzt, einer von ihnen steht in einem allgemeinen Register und der andere im Hauptspeicher. Diese Überlappung von Operationen kann selbstverständlich die Verarbeitung wesentlich beschleunigen.

3.9. Die Steuerung von Computern

Unter diesem Thema wollen wir uns mit der Frage beschäftigen, wie einem Computer mitgeteilt wird, welche Operation er als nächste ausführen soll. Um einem Computer mitzuteilen, was er tun soll, müssen wir ihm eine Instruktion vorgeben. Mit allgemeinen Worten ausgedrückt, könnten Instruktionen etwa wie folgt lauten:

a) Übertrage ins allgemeine Register 7 das Wort mit der Adresse 003140
b) Addiere zum Inhalt des allgemeinen Registers 7 das Wort mit der Adresse 004132

Praktisch ist ein solches Vorgehen undurchführbar, würde es doch voraussetzen, daß ein Computer die deutsche Umgangssprache versteht und ihre Befehle ausführen kann. Infolgedessen müssen wir Computern ihre Befehle in einem bestimmten leicht durchschaubaren Code geben. Die beiden obigen Instruktionen könnten wir etwa wie folgt simplifizieren:

a) 58 7 3140
b) 5A 7 4132

Die beiden ersten Sedezimalziffern sollen in verschlüsselter Form die auszuführende Operationsart kennzeichnen (Operationsschlüssel). Die beiden übrigen Angaben haben wir in derselben Reihenfolge und Form, freilich stark abgekürzt, niedergeschrieben, wie wir sie zuerst mit deutschen Worten prägten. Die soeben eingeführte Darstellungsweise erspart uns Schreibarbeit. Wenn wir jedoch diese Instruktionen der IBM /370 präsentieren wollen, so müssen wir sie in einer Form anbieten, die die IBM /370 auch versteht. Wie die meisten Computer verlangt die IBM /370, daß die ihr zugänglich zu machenden Instruktionen in Form numerisch verschlüsselter Mengen dargeboten werden.

Etwa im Jahre 1945 wurde entdeckt, daß Computer wirkungsvoller genutzt werden könnten, wenn die verschlüsselten Instruktionen zusammen mit den Daten, mit denen sie Operationen ausführen sollten, im gleichen Speicher gespeichert würden. Zum Beispiel könnten wir die Übertragungsinstruktion von a) auf den Speicherplätzen mit den Adressen 002000 bis 002003 speichern, die Addition von b) auf den angrenzenden Speicherplätzen mit den Adressen 002004 bis 002007. Würde nun der Computer veranlaßt, die Instruktion auszuführen, die auf dem Speicherplatz mit der Adresse 002000 beginnt, so könnte er anschließend mit der auf 002004 beginnenden Instruktion fortfahren, dann mit der auf 002008 beginnenden usw. Auf dieser Basis fußend, könnte ein umfangreiches, aus einer bestimmten Instruktionsmenge bestehendes Programm ablaufen.

Nehmen wir einmal an, wir wollen ein Programm schreiben, das zunächst eine Lochkarte liest, anschließend zwei Zahlen addiert, die in bestimmten Kartenfeldern abgelocht sind und zum Schluß eine neue Karte stanzt. Wenn für mehrere tausend Karten dieselben Operationen durchgeführt werden sollen, so können wir ein für diesen Fall geeignetes, aus einzelnen Instruktionen bestehendes Programm der IBM /370 zuleiten und es ablaufen lassen. Nach dem Stanzen einer neuen Karte hängen wir einfach eine Instruktion an, die dem Computer sagt, daß er zum Anfang der Instruktionsfolge zurückgehen und mit der ersten Instruktion erneut starten soll. Der Computer würde auf diesem Wege solange fortfahren, bis es keine zu lesenden Lochkarten mehr gibt.

Liegt eine solche Instruktionsfolge im Hauptspeicher vor, so sparen wir die Zeit an, die sonst notwendig wäre, um jede einzelne Instruktion vor ihrer Ausführung erst einzulesen. Später werden wir erkennen, daß der soeben dargestellte Aspekt der Geschwindigkeitserhöhung nur einer der Vorteile ist, die sich daraus ergeben, daß das Programm selbst im Hauptspeicher des Computers gespeichert ist.

3.10. Das Datenverarbeitungssystem IBM /370

Zu Beginn dieses Kapitels stellten wir fest, daß ein Computer vier verschiedene Gruppen von Operationen ausführen können muß:

1) Eingabe/Ausgabe, kurz EA
2) Speichern
3) Arithmetische Operationen
4) Steuerungsoperationen

Physisch gesehen, besteht ein Datenverarbeitungssystem IBM /370 aus drei verschiedenen Gerätetypen:

1) Die Speichereinheit enthält den Hauptspeicher (bipolare MOS-Schaltkreise oder MOSFET-Schaltkreise).

2) Die Zentraleinheit [1] enthält das Rechenwerk und das Steuerwerk, d.h. die Schaltkreise zur Durchführung der arithmetischen Operationen und die Schaltkreise zur Auslösung der Steuerfunktionen. Die Register sind ebenfalls in der Zentraleinheit untergebracht. Bei einigen Modellen der IBM /370 ist die Speichereinheit ebenfalls Bestandteil der Zentraleinheit, doch werden auch in diesem Fall Zentraleinheit und Speichereinheit als zwei getrennte logische Einheiten angesehen.

3) Die Eingabe/Ausgabe-Geräte [2] führen physisch die Eingabe/Ausgabe-Operationen aus. Bei einem System IBM /360 brauchen nur einige wenige dieser Geräte vorhanden zu sein, es können aber auch eine beträchtliche Anzahl derartiger Geräte angeschlossen sein. Einige der wichtigsten Typen von Eingabe/Ausgabe-Geräten sollen jetzt aufgeführt werden:

   a) Lochkartenleser und Lochkartenstanzer
   Diese Geräte besorgen das Lesen von in Lochkarten gestanzten Informationen sowie das Stanzen des Inhalts bestimmter Hauptspeicherteile in Lochkarten. Einige Sonderausführungen dieser Geräte ermöglichen darüberhinaus neben dem Stanzen auch das Bedrucken von Lochkarten.

   b) Zeilendrucker
   Diese unterscheiden sich von den Schreibmaschinen dadurch, daß sie eine ganze Zeile in einem Arbeitsgang bedrucken, und zwar mit einer Geschwindigkeit von bis zu 2500 Zeilen pro Minute. Eine Zeile kann dabei zwischen 120 bis 160 Druckstellen umfassen.

   c) Magnetbandgeräte
   Diese Geräte sind größere, schnellere und zuverlässigere Versionen unserer gewöhnlichen Tonbandgeräte, doch funktionieren sie, im Prinzip gesehen, in der gleichen Weise. Im Hauptspeicher stehende Informationen können auf Magnetbänder aufgezeichnet werden; von diesen selbst wiederum können sie zurückgespielt werden und somit in den Hauptspeicher eingelesen werden. Magnetbandgeräte sind Speichergeräte mit relativ niedrigen Kosten.

   d) Magnetplattenspeichergeräte
   Diese Geräte weisen rein äußerlich eine bescheidene Ähnlichkeit mit Mehrfachplattenspielern auf. Auf den Platten, die mit einer dünnen magnetisierbaren Schicht überzogen sind, werden die Informationen aufgezeichnet, die vom Haupt-

1) Im angloamerikanischen Sprachraum al Central Processing Unit, abgekürzt CPU, bezeichnet.
2) Im angloamerikanischen Sprachraum als Input/Output-Units, abgekürzt I/O-Units, bezeichnet.

speicher her übertragen werden (in ähnlicher Weise, wie es bei einem Diktiergerät geschieht). Später können dann die Informationen in den Hauptspeicher zurückgespielt werden; man sagt, sie werden in den Hauptspeicher gelesen.

Neben diesen hier erwähnten Geräten existieren noch eine Fülle weiterer, über die wir zu gegebener Zeit sprechen werden.

Bezüglich der Speicherungs- bzw. Aufzeichnungsverfahren sollen noch einige Klarstellungen erfolgen, um jegliche Mißverständnisse von vornherein auszuräumen. Informationen sind auf Magnetbändern und Magnetplatten Byte für Byte genau so aufgezeichnet, wie die Bytes im Hauptspeicher stehen. Auf Karten stehende Informationen werden durch die Kartenleser vom Hollerithcode (Kartencode) in den EBCDI-Code übersetzt. Bei der Ausgabe auf Karten erfolgt eine Übersetzung vom EBCDI-Code in den Kartencode.

Aufgabengruppe 7

1) Das Modell 115 der Computerfamilie IBM /370 liest in 480 ns zwei Bytes aus dem Hauptspeicher ein, das Modell 195 hingegen sechzehn Doppelwörter in 756 ns. Wieviele Doppelwörter kann das Modell 195 in der gleichen Zeit lesen, in der das Modell 115 ein Doppelwort liest? Beantworte die gleiche Frage auch für das Modell 155, das zwei Doppelwörter in 2,070 ns liest.

2) Schreibe die alphamerische Nachricht nieder, die durch die nachstehende Folge von Sedezimalziffern verschlüsselt ist! (EBCDI-Code)

   a) D5 D6 E6 40 C9 E2 40 E3 C8 C5 40 E3 C9 D4 C5
   b) C5 40 D7 D3 E4 D9 C9 C2 E4 E2 40 E4 D5 E4 D4 4B
   c) F1 F2 61 F3 40 7E 40 F4 4b
   d) C1 93 93 40 C7 81 A4 93 40 89 A2 40 84 89 A5 89 84 85 84 40 99 95 A3 96 40 F3 40 97 81 99 A3 A2 4B

3) Schreibe die folgenden Texte verschlüsselt mit Sedezimalziffern nieder! (EBCDI-Code)

   a) TAUGHT TO THE TUNE OF THE HICKORY STICK
   b) If 7 + X/5=9, then X=10.
   c) Lernen und leisten befriedigt am meisten.
   d) WENN 4(2X-1,5)=8, DANN X=1,75.

4) Welche der nachfolgenden Speicherplätze geben eine erlaubte Wortbegrenzung an?
   a) 000008 b) 00000C c) 0000A5 d) 001200 e) 001896 f) 001899 g) 00190E
   h) EC3452

5) Ausgehend von Aufgabe 4) soll folgende Frage beantwortet werden:
   Welche Sedezimalziffern dürfen in der Einerstelle einer Adresse auftreten, damit diese als Begrenzung eines Wortes genommen werden kann?

6) Beantworte dieselbe Frage wie in Aufgabe 5) für die Adressen von Halbwörtern und von Doppelwörtern!

7) Wenn ein Zeilendrucker mit einer Geschwindigkeit von 2500 Zeilen pro Minute arbeitet, wieviele Millisekunden benötigt er dann, um eine Zeile zu drucken?

8) Wenn ein Kartenleser 1000 Karten pro Minute spaltenweise seriell liest (d.h. erst Spalte 1, dann Spalte 2 usw.), welche Zeit in Millisekunden benötigt er dann, um eine einzige Spalte zu lesen?

4. Instruktionen

4.1. Die Instruktionen der IBM/370

Die IBM /370 ist eine speicherprogrammierte Datenverarbeitungsanlage (speicherprogrammierte Rechenanlage). Darunter versteht man eine Anlage, bei der die auszuführenden Instruktionen ebenso im Hauptspeicher gespeichert sind wie die Zahlen oder Daten, die sie verarbeiten sollen. Im allgemeinen sind die Instruktionen bei der IBM /370 wie folgt aufgebaut: Sie beginnen mit einem Operationsschlüssel (im folgenden kurz als OP bezeichnet); diesem schließen sich dann zwei Adressen an, die die Daten oder Operanden angeben, mit denen die betreffende Operation durchgeführt werden soll. Infolgedessen sieht eine Instruktion schematisch wie folgt aus:

Operations-schlüssel (OP)	Adresse des 1. Operanden (Operand 1)	Adresse des 2. Operanden (Operand 2)

Abb. 4.1. Instruktionsaufbau

Die Daten können sich entweder im Hauptspeicher oder in den Registern befinden.

Der Operationsschlüssel wird durch zwei Sedezimalziffern ausgedrückt, er belegt also ein Byte. Somit sind maximal 256 verschiedene Operationen durchführbar, jedoch sind nur 166 wirklich definiert. Um sich auf einen Operanden zu beziehen, der sich in einem Register befindet, genügt die Angabe einer einzigen Sedezimalziffer, da gerade 16 Register vorhanden sind. Befinden sich beide Operanden in Registern, so nimmt eine Instruktion die folgende Form an:

Byte	0	1	
Inhalt des Bytes	Operations-schlüssel (OP)	R_1	R_2

Abb. 4.2. Instruktion im RR-Format

Hierbei bedeuten R_1 und R_2 zwei Register, in denen sich der erste bzw. der zweite Operand befinden. Addieren wir die zwei Sedezimalziffern dazu, die zur Festlegung des OP erforderlich sind, so finden wir, daß eine solche Instruktion gerade vier Sedezimalziffern erfordert, d.h. somit, daß sie ein Halbwort, d.h. zwei Bytes belegt. Dieses Instruktionsformat heißt kurz RR-Format (Register zu Register).

Die Angabe einer Adresse für Operanden, die sich im Hauptspeicher befinden, ist komplizierter. Da immerhin rd. 16 Millionen Bytes adressiert werden müssen, sind sechs Sedezimalziffern erforderlich, um eine volle Hauptspeicheradresse anzugeben. Um Speicherplatz zu sparen, der trotz aller technischen Fortschritte relativ teuer ist,

wollen wir aber die Instruktionen so kurz wie möglich halten. Daher werden beim System IBM /370 Hauptspeicheradressen in der Form

X B DDD

oder in der Form

X DDD

angegeben.

Hierbei bedeuten X und B Sedezimalziffern, die entweder gleich 0 sind oder die sich auf eines der allgemeinen Register 1 bis 15 beziehen; DDD ist eine dreiziffrige Sedezimalzahl, welche Distanzadresse [1] (oder Displacement) genannt wird. Das durch X spezifizierte Register heißt Indexregister und sein Inhalt Index, während das durch B spezifizierte Register Basisregister genannt und sein Inhalt als Basisadresse bezeichnet wird. Ist B=0, so wird das allgemeine Register 0 nicht für die Adreßbildung herangezogen; deshalb sprechen wir hier davon, daß die Basisadresse gleich Null ist. Ebenso wird auch keine Indizierung durchgeführt, wenn X=0 ist.

Die Hauptspeicheradresse wird dadurch gebildet, daß die Distanzadresse zur Basisadresse addiert wird. Ist ein Indexregister ungleich Null angegeben, so wird außerdem noch der Index addiert. Die auf diese Weise erhaltene Summe ist die effektive Adresse; wir wollen sie mit E bezeichnen. In symbolischer Form können wir somit schreiben:

$$E = D + (B) + (X)$$

Der Kürze wegen schreiben wir hier für das Displacement einfach D statt DDD, während die Klammern bedeuten sollen, daß der Inhalt der entsprechenden Register gemeint ist. Bei der Durchführung der Additionen werden alle Zahlen als ganze Dualzahlen angesehen. Ergibt sich bei der Summenbildung ein Wert, der größer ist als der maximal für Adressen zulässige Wert, d.h. daß er nicht in drei Bytes untergebracht werden kann, dann werden nur die sechs Sedezimalziffern niedrigster Ordnung als Adresse gewertet.

Eine recht häufig benutzte Art von Instruktionen ist im sogenannten RX-Format aufgebaut. Die Abkürzung RX kommt daher, daß es sich um Instruktionen handelt, deren erster Operand in einem Register (R) steht und deren zweiter Operand sich im Hauptspeicher befindet und dort durch eine indizierte Adresse (X) angesprochen werden kann, deshalb also die Kurzbezeichnung RX. Eine im RX-Format vorliegende Instruktion besitzt folgenden Aufbau:

Byte	0	1		2	3
Inhalt des Bytes	Operations-schlüssel (OP)	R_1	X_2	B_2	D_2

Abb. 4.3. Instruktion im RX-Format

Hierbei bedeutet R_1 die Registeradresse des ersten Operanden, während aus X_2, B_2 und D_2 die Hauptspeicheradresse des zweiten Operanden gebildet wird.

1) Statt „Distanzadresse" findet man hin und wieder auch den Begriff „relative Adresse".

Neben diesen beiden Instruktionsformaten verwendet die IBM /370 auch noch andere Instruktionsformate, die wir zu gegebener Zeit besprechen werden.

4.2. Daten in Festpunktdarstellung

Instruktionen können nach verschiedenen Gesichtspunkten in Klassen eingeteilt werden, u. a. nach den Datentypen, mit denen sie arbeiten, aber auch nach den Instruktionsformaten. Bleiben wir bei der ersten Einteilung und behandeln zunächst Instruktionen, die die sogenannten Festpunktdaten [1] verarbeiten können. Unter einer Festpunktzahl wollen wir ganze Dualzahlen verstehen, die ein Halbwort oder ein Wort belegen. Auf der Bitstelle 0 des Halbwortes bzw. des Vollwortes ist das Vorzeichen der Daten festgehalten und zwar 0 für positive Daten und 1 für negative Daten. Der Dualpunkt wird, wie schon gesagt, rechts von der letzten Bitstelle angenommen. Festpunktdaten sehen also wie folgt aus:

±xxxxxxxxxxxxxxx. (für Halbwörter) bzw.
±xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx. (für Wörter)

Bei dieser Darstellung sind die Dualziffern durch x angedeutet worden. Aus Bequemlichkeitsgründen wollen wir meist Sedezimalziffern beim Niederschreiben von Festpunktdaten benutzen. Betrachten wir deshalb einige Beispiele:

Dezimalzahl	Dualzahl (Wort)	Dualzahl (Halbwort)
67 345	00010711	—
3 104	00000C20	0C20
24 832	00006100	6100
133	00000085	0085
7	00000007	0007
2 147 483 647	7FFFFFFF	—
32 767	00007FFF	7FFF
0	00000000	0000

Negative Zahlen werden im Zweierkomplement dargestellt.

Dezimalzahl	Dualzahl (Wort)	Dualzahl (Halbwort)
-67 345	FFFEF8EF	—
-3 104	FFFFF3E0	F3E0
-24 832	EFFF8E00	8E00
- 133	FFFFFF7A	FF7A
-7	FFFFFFF9	FFF9
-2 147 483 647	80000001	—
-2 147 483 648	80000000	—
-32 768	FFFF8000	8000
-32 767	FFFF8001	8001

1) Auch als Festkommadaten bezeichnet; die Darstellungsart heißt mitunter auch Festkommadarstellung.

Man beachte, daß die kleinste darstellbare negative Zahl, absolut genommen, um 1 größer ist als die größte darstellbare positive Zahl. Es sollte aber auch vermerkt werden, daß es bei der Erweiterung eines Halbwortes zu einem Wort nur notwendig ist, das Vorzeichenbit in die zusätzlichen 16 Bitstellen einzuschleusen.

Da die Stellenzahl einer Festpunktzahl einschließlich der Vorzeichenstelle 16 bzw. 32 Stellen nicht überschreitet, kann eine solche Zahl in einem allgemeinen Register untergebracht werden. Somit ist es außerordentlich sinnvoll, wenn man die allgemeinen Register dazu benutzt, um arithmetische Operationen mit Festpunktdaten durchzuführen.

4.3. Instruktionen zur Verarbeitung von Daten in Festpunktdarstellung

a) den Namen der Instruktion in Deutsch und Englisch, z.B. Addieren - Add.
b) den Operationsschlüssel, z.B. 5A
c) eine Gedächtnishilfe für den Operationsschlüssel (mnemonischer Operationsschlüssel), z.B. A (für Addieren)
d) das Format der Instruktion, z.B. RX
e) eine Erläuterung der Instruktion.

Durch einen Pfeil wollen wir auf den Platz verweisen, auf dem das Resultat der durch die Instruktion bewirkten Operation untergebracht wird. Durch das Zeichen ·/· wollen wir vermerken, daß zwei Daten miteinander verglichen werden, ohne daß das Vergleichsergebnis irgendwo gespeichert wird.

Da es außerordentlich schwierig ist, sich die echten Operationsschlüssel zu merken, wollen wir uns auf eine Instruktion stets durch Angabe ihres mnemonischen Operationsschlüssels beziehen.

Manch einer würde erwarten, daß wir unsere Besprechung mit den Eingabe/Ausgabe-Instruktionen [1] beginnen, da es augenscheinlich zuerst einmal notwendig ist, die Daten und die Instruktionen in den Hauptspeicher zu lesen, bevor überhaupt eine Datenverarbeitung einsetzen kann. Um jedoch die Eingabe/Ausgabe-Operationen [2] verstehen zu können, müssen wir uns zunächst einige Kenntnisse darüber aneignen, wie die anderen Instruktionen arbeiten. Deshalb setzen wir für dieses Kapitel voraus, daß es eine Methode gibt, mit deren Hilfe die Daten in den Hauptspeicher gebracht werden können und mit der ebenso wieder Daten aus dem Hauptspeicher ausgegeben werden können.

1) Kurzbezeichnung: EA-Instruktionen.
2) Kurzbezeichnung: EA-Operationen.

Wir wollen mit der Besprechung einiger Instruktionen [1] beginnen.

1. Ladeinstruktionen

a)

Laden — Load	58	L	RX

58	R_1	X_2	B_2	D_2

Ablauf der Operation: $(E_2) \rightarrow (R_1)$

Der zweite Operand wird in das durch den ersten Operanden spezifizierte Register übertragen. Der vorhergehende Inhalt des Registers R_1 geht verloren. Der zweite Operand bleibt unverändert. [Beachte, daß durch die Klammern angezeigt wird, daß der Inhalt des Speicherplatzes genommen wird, der mit der effektiven Adresse E_2 angesprochen wird. Somit bedeutet (E_2) die Zahl, die auf dem Speicherplatz mit der effektiven Adresse E_2 gespeichert ist.]

b)

Umladen Register — Load	18	LR	RR

18	R1	R2

$(R_2) \rightarrow (R1)$

Es läuft die gleiche Operation wie bei L ab, nur daß der zweite Operand ein Register ist.

Beispiele:

Inhalte vor Ausführung der Instruktionen:

Register 3	0000 0004
Register 4	0000 0300
Register 5	0000 0021
Speicherplätze 000 304	
bis 000 307	0045 6BF3

1. Instruktion: LR 18 5 3

Inhalte nach Ausführung dieser Instruktion:

Register 3 u. 4	unverändert
Register 5	0000 0004
Speicherplätze 000 304	
bis 000 307	unverändert

1) Bei den Besprechungen der Instruktionen wird grundsätzlich folgende Reihenfolge eingehalten: Deutsche Bezeichnung, Englische Bezeichnung, echter Operationsschlüssel, mnemonischer Operationsschlüssel, Instruktionsformat.

2. Instruktion: L 58 5 0 4 004

Inhalte nach Ausführung dieser Instruktion:

Register 3 u. 4	unverändert
Register 5	0045 6BF3
Speicherplätze 000 304 bis 000 307	unverändert

2. Additionsinstruktionen

a)

Addieren — Add	5A	A	RX

$(R_1) + (E_2) \rightarrow (R_1)$

b)

Addieren Register — Add	1A	AR	RR

$(R_1) + (R_2) \rightarrow (R_1)$

Der zweite Operand wird auf den ersten Operanden addiert (A wie auch AR). Die Summe ersetzt den ersten Operanden, der zweite Operand bleibt unverändert.

Beispiele:

Inhalte vor Ausführung der Instruktionen:

Register 3	0000 0004
Register 4	0000 0300
Register 5	0067 891B
Speicherplätze 000304 bis 000307	0045 6BF3

1. Instruktion AR 1A 34

Inhalte nach Ausführung dieser Instruktion:

Register 3	0000 0304
Register 4	unverändert

2. Instruktion: A 5A 5 0 4 004

Inhalte nach Ausführung dieser Instruktion:

Register 5	00AC F50E
Speicherplätze 000304 bis 000307	unverändert

3. Subtraktionsinstruktionen

a)

Subtrahieren — Subtract	5B	S	RX

$(R_1) - (E_2) \rightarrow (R_1)$

b)

Subtrahieren Register — Subtract	1B	SR	RR

$(R_1) - (R_2) \rightarrow (R_1)$

Bei beiden Instruktionen wird der erste Operand um den zweiten Operanden vermindert; die Differenz ersetzt den ersten Operanden. Der zweite Operand verbleibt unverändert. Tatsächlich bildet der Computer intern das Zweierkomplement des zweiten Operanden und addiert dieses auf den ersten Operanden.

Beispiele:

Inhalte vor Ausführung der Instruktionen:

Register 3	0000 0034
Register 5	0067 891B

1. Instruktion: SR 1B 53

Inhalte nach Ausführung dieser Instruktion:

Register 3	unverändert
Register 5	0067 88E7

2. Instruktion: SR 1B 55

Inhalte nach Ausführung dieser Instruktion:

Register 3	unverändert
Register 5	0000 0000

4. Speicherungsinstruktionen

Speichern Wort — Store full word	50	ST	RX

$(R_1) \rightarrow (E_2)$

Der Inhalt des durch den ersten Operanden angegebenen Registers wird auf den Speicherplatz übertragen, der durch den zweiten Operanden spezifiziert wird. Diese Instruktion bewirkt also gerade das Gegenteil der Instruktion Laden; hier wird der Inhalt eines Registers unverändert in ein Hauptspeicherwort übertragen. Für diese Instruktionsart gibt es kein RR-Format.

Beispiel:

Inhalte vor Ausführung der Instruktion:

Register 4	0000 3000
Register 7	FF58 F399
Speicherplätze 00300C bis 00300F	7777 8888

Instruktion: ST 50 7 0 4 00C

Inhalte nach Ausführung der Instruktion:

Register 4	unverändert
Register 7	unverändert
Speicherplätze 00300C bis 00300F	FF58 F399

5. Multiplikationsinstruktionen

5) a)

Multiplizieren — Multiply	5C	M	RX

$(R_1 + 1) \cdot (R_2) \rightarrow (R_1, R_1 + 1)$

b)

Multiplizieren Register — Multiply	1C	MR	RR

$(R_1 + 1) \cdot (E_2) \rightarrow (R_1, R_1 + 1)$

Multiplikationsinstruktionen ist es erforderlich, daß als Register R_1 ein Register mit gerader Ordnungszahl angegeben wird, also 2 oder 4 oder usw. Die höheren Stellen des Produktes werden im Register R_1 gespeichert, die niedrigeren Stellen im anschließenden Register, also im Register $R_1 + 1$. Das Vorzeichen des Produktes wird im Register R_1 gespeichert, es erscheint nicht gleichzeitig im Register $R_1 + 1$, denn alle 32 Bitstellen dieses Registers werden von den 32 letzten Dualziffern des Produktes belegt.

Beispiele:

Inhalte vor Ausführung der Instruktionen:

Register 0	FFFF 1378
Register 1	0200 E001
Register 2	0000 0086
Register 3	0000 0034
Register 4	0000 3000
Speicherplätze 003010 bis 003013	FFFF FFF4

1. Instruktion: M 5C 2 0 4 010

Inhalte nach Ausführung dieser Instruktion:

Register 2	FFFF FFFF
Register 3	FFFF FD90
Register 0, 1 und 4	unverändert
Speicherplätze 003010 bis 003013	unverändert

2. Instruktion: MR 1C 0 1

Inhalte nach Ausführung dieser Instruktion:

Register 0	0004 0380
Register 1	C801 C001
Register 2, 3 und 4	unverändert
Speicherplätze 003010 bis 003013	unverändert

3. Instruktion: MR 1C 2 2

Inhalte nach Ausführung dieser Instruktion:

Register 2	0000 0000
Register 3	0000 1B38
Register 0, 1 und 4	unverändert
Speicherplätze 003010 bis 003013	unverändert

Man beachte, daß die vordere Bitstelle des Registers mit ungerader Ordnungszahl kein Vorzeichen anzeigt.

6. Divisionsinstruktionen

a)

Dividieren — Divide	5D	D	RX

$(R_1, R_1+1) : (E_2) \rightarrow$ { Quotient in R_1+1 / Rest in R_1

b)

Dividieren Register — Divide	1D	DR	RR

$(R_1, R_1+1) : (R_2) \rightarrow$ { Quotient in R_1+1 / Rest in R_1

Bei beiden Instruktionen muß mit R_1 ein Register mit gerader Ordnungszahl angegeben werden. Der Dividend muß die 64 Bitstellen des Registerpaares R_1 und R_1+1 belegen. Der Divisor steht entweder auf den durch E_2 spezifizierten Speicherplätzen oder im Register R_2. Dividend und Divisor werden als ganze Zahlen betrachtet. Bei Ausführung der Division bildet sich der Rest im geraden Register R_1, der Quotient dagegen im darauffolgenden ungeraden Register R_1+1. Das Vorzeichen des Quotienten wird nach den algebraischen Regeln bestimmt. Der Rest hat stets das gleiche Vorzeichen wie der Divisor, falls er nicht gleich Null ist. Man muß unbedingt darauf achten, daß der Quotient nicht größer werden kann als die größte 31 Bitstellen belegende Zahl (die 32. Bitstelle muß ja für das Vorzeichen reserviert bleiben). Wird diese Voraussetzung nicht eingehalten, so wird die Division nicht ausgeführt; außerdem tritt eine Fehlerbedingung auf.

Beispiele:

Inhalte vor Ausführung der Instruktionen:

Register 0	0375 8142
Register 1	1701 0271
Register 8	FFFF FFFF
Register 9	FFFF 5678
Register 10	0000 0034

1. Instruktion DR 1D 01

Inhalte nach Ausführung dieser Instruktion:

Register 0	0E07 D2EA
Register 1	267E 5D77
Register 8, 9 und 10	unverändert

2. Instruktion DR 1D 8A

Inhalte nach Ausführung dieser Instruktion:

Register 8	FFFF FFE0
Register 9	FFFF FCBE
Register 0, 1 und 10	unverändert

Aufgabengruppe 8

1) Bekannt sind die Inhalte folgender Register und Speicherplätze:

Register	12	0000 0008
Register	13	0000 1000
Speicherplätze	001 000	
bis	001 003	0000 0007
Speicherplätze	001 004	
bis	001 007	0000 003A
Speicherplätze	001 008	
bis	001 00B	FFFF 17A0
Speicherplätze	001 00C	
bis	001 00F	0017 8306

Gesucht ist der Inhalt des Registers 12 nach Ausführung der folgenden Instruktionen:

a) L 58 C 0 D 008
b) A 5A C 0 D 000
c) LR 18 C C
d) S 5B C C D 004
e) SR 1B C D
f) AR 1A C C
g) M 5C C 0 D 00C
h) SR 1B C C
i) D 5D C 0 D 00C
j) D 5D C 0 D 004
k) MR 1C C C
l) LR 18 D C

2) Unter Berücksichtigung der in Aufgabe 1) gegebenen Daten sind Instruktionen anzugeben, die folgende Probleme lösen:

a) Löschen des Registers 11 (auf Null setzen).
b) Der Inhalt von Register 11 soll 0000 1000 sein.
c) Der Inhalt von Register 11 soll FFFF 17A0 sein.
d) Register 12 soll auf 0000 000F gesetzt werden.
e) Register 12 soll den Inhalt 0000 0010 erhalten.
f) Register 13 soll auf 0100 0000 gesetzt werden.

3) Unter Berücksichtigung der bei Aufgabe 1) gegebenen Daten ist eine Instruktion anzugeben, die die Speicherplätze 001 004 bis 001 007 auf folgende Werte setzt:

   a) 0000 0008
   b) 0000 1000

4) Unter Berücksichtigung der bei Aufgabe 1) gegebenen Daten sind zwei Instruktionen niederzuschreiben, die die Speicherplätze 001 004 bis 001 007 auf folgende Werte setzen:

   a) 0000 0042
   b) 0000 0007
   c) FFFF 17A8
   d) 0000 1008

4.4. Verzweigungen

Die auszuführenden Instruktionen müssen selbst irgendwo im Hauptspeicher gespeichert sein. Üblicherweise speichert man Instruktionen und Daten in verschiedenen Hauptspeichergebieten, doch besteht dafür keinerlei Notwendigkeit. Wir wollen jetzt ein Programm verfolgen, in das zwei Festpunktzahlen eingehen; die eine ist auf den Speicherplätzen 003 100 bis 003 103 gespeichert, die andere auf den Speicherplätzen 003 104 bis 003 107. Im Programm soll die Summe dieser beiden Zahlen berechnet werden und auf den Speicherplätzen 003 108 bis 003 10B gespeichert werden. Wir können dazu folgende Instruktionsfolge verwenden:

Speicherplätze	Instruktionen	
	Mnemonischer Oper.-Schl.	Instruktion im Maschinencode
004 000 bis 004 003	L	58 2 0 4 000
004 004 bis 004 007	A	5A 2 0 4 004
004 008 bis 004 00B	ST	50 2 0 4 008

Voraussetzung: Im Register 4 ist der Wert 0000 3100 gespeichert.

Abb. 4.4. Instruktionsfolge

Hegen wir den Wunsch zurückzugehen und die Instruktionen zu wiederholen, so benötigen wir wir eine Instruktion, die dem Computer mitteilt, daß er die Ausführung der Instruktionen von einem Speicherplatz aus starten soll, der nicht gleich dem nächstfolgenden Speicherplatz ist. Solche Instruktionen werden als Verzweigungsinstruktionen bezeichnet. Die wichtigste aller Verzweigungsinstruktionen ist die folgende:

Verzweigen bei Bedingung — Branch on Condition	47	BC	RX

47	M_1	X_2	B_2	D_2

Ist irgendeine der in M_1 festgehaltenen Bedingungen erfüllt, so wird als nächste Instruktion diejenige ausgeführt, die auf einem Speicherplatz beginnt, dessen Adresse gleich E_2

ist. Ist keine Bedingung erfüllt, so wird die nächstfolgende Instruktion ausgeführt. Bei dieser Instruktion stellt der erste Operand keine wirkliche Adresse dar. Stattdessen halten seine vier Bitstellen die „Verzweigungsbedingung" fest. Mit dieser Verzweigungsbedingung wird ein Bedingungsschlüssel des Computers geprüft. Ist die Prüfung erfolgreich verlaufen, so wird die Verzweigung ausgeführt. Die vier Bitstellen, die die Verzweigungsbedingung festhalten, werden häufig als Maskenbitstellen für die Verzweigung bezeichnet.

Der Bedingungsschlüssel kann vier mögliche Werte aufweisen: 0, 1, 2 und 3. Diese Werte werden als Resultat bestimmter Instruktionen gesetzt. Bleiben wir z.B. bei den Additionsinstruktionen: Der Bedingungsschlüssel erhält den Wert 0, wenn die Summe gleich Null ist, ihm wird der Wert 1 zugewiesen, wenn die Summe negativ ist, und der Wert 2, wenn die Summe positiv ist. Die vier Bitstellen 0, 1, 2 und 3 von M_1 sind nun auf 1 zu setzen, wenn der Programmierer wünscht, daß eine Verzweigung auf Grund des entsprechenden Wertes des Bedingungsschlüssels erfolgt. Setzt man M_1 auf 0010, so erfolgt die Verzweigung, wenn der Bedingungsschlüssel den Wert 2 besitzt, setzt man M_1 auf 1010, so wird eine Verzweigung hervorgerufen, wenn der Bedingungsschlüssel entweder gleich 0 oder gleich 2 ist; M_1 auf 0111 gesetzt bewirkt schließlich eine Verzweigung, wenn der Bedingungsschlüssel nicht gleich 0 ist. Aus diesen wenigen Beispielen kann man bereits ersehen, daß keine Verzweigung erfolgt, wenn M_1 auf 0000 steht. Eine BC-Instruktion mit einem derartigen Aufbau ihrer Maskenbitstellen hat demnach keine Operation zur Folge; man nennt sie deshalb meist eine Nop-Instruktion [1]. Steht dagegen M_1 auf dem Wert 1111, so erfolgt immer eine Verzweigung; eine solche BC-Instruktion wird deshalb als unbedingte Verzweigungsinstruktion bezeichnet.

Beispiele:

Zustand vor Ausführung der Instruktionen:

Bedingungsschlüssel	2
Inhalt von Register 4	0000 3100
Inhalt von Register 5	0000 4000

1. Instruktion auf den Speicherplätzen 004 00C bis 004 00F:

BC 47 4 0 5 000

Ergebnis dieser Instruktion:

Keine Verzweigung; die nächste auszuführende Instruktion beginnt auf dem Speicherplatz 004 010.

2. Instruktion auf den Speicherplätzen 004 00C bis 004 00F:

BC 47 6 0 5 000

Ergebnis dieser Instruktion:

Verzweigung; die nächste auszuführende Instruktion beginnt auf dem Speicherplatz 4000, denn die effektive Adresse E_2 ist gleich dem Inhalt von Register 5.

1) Nop — No Operation.

Durch die nachfolgenden bisher behandelten Instruktionen wird der Bedingungsschlüssel neu gesetzt:

1) Addieren a) A (RX-Format)
   b) AR (RR-Format)

2) Subtrahieren a) S (RX-Format)
   b) SR (RR-Format)

Durch diese Instruktionen wird ferner der Bedingungsschlüssel auf 3 gesetzt, wenn es zu einem Überlauf kommt, d.h., daß das Resultat so groß ist, daß es vom Ergebnisregister nicht mehr aufgenommen werden kann. Ist das nicht der Fall, so wird der Bedingungsschlüssel auf folgende Werte gesetzt: a) auf 0, wenn der Ergebnis gleich 0 ist, b) auf 1, wenn das Ergebnis negativ ist, c) auf 2, wenn das Ergebnis positiv ist.

Von der Ausführung einer der Instruktionen Laden, Speichern, Multiplizieren und Dividieren wird der Bedingungsschlüssel nicht berührt, d.h. sein bisheriger Wert bleibt erhalten.

In Verbindung mit dem Bedingungsschlüssel sind einige andere Instruktionen äußerst wertvoll:

1. Vergleichsinstruktionen

a)

Vergleichen — Compare	59	C	RX

$(R_1)-(E_2) \rightarrow \cdot/\cdot$

$(R_1)-(E_2)$ wird gebildet, ohne daß das Ergebnis gespeichert wird (siehe auch Abschnitt 4.3.).

b)

Vergleichen Register — Compare	19	CR	RR

$(R_1)-(R_2) \rightarrow \cdot/\cdot$

Wie bei den Subtraktionsinstruktionen wird der zweite Operand vom ersten Operanden subtrahiert; die Differenz wird jedoch nicht festgehalten. Der Bedingungsschlüssel wird in genau der gleichen Weise gesetzt wie bei den Subtraktionsinstruktionen. Der Inhalt des ersten Registers, d.h. (R_1), wird also nicht verändert.

2. Instruktionen zum Laden und Setzen des Bedingungsschlüssels

a)

Umladen und Prüfen — Load and Test	12	LTR	RR

$(R_2) \rightarrow (R_1)$

Bei Ausführung dieser Instruktion läuft grundsätzlich die gleiche Operation ab wie bei der Instruktion LR, nur daß außerdem auch der Bedingungsschlüssel auf 0 oder auf 1 oder auf 2 gesetzt wird, je nachdem, ob die ins Register R_1 geladene Zahl gleich Null oder negativ oder positiv ist.

Von der im Register R_2 stehenden Zahl wird das Zweierkomplement gebildet und in das Register R_1 geladen. Der Bedingungsschlüssel wird nach den bekannten Regeln gesetzt, je nachdem, ob die ins Register R_1 geladene Zahl gleich 0 oder negativ oder positiv ist. Der Bedingungsschlüssel kann außerdem auch auf den Wert 3 gesetzt werden, dann nämlich, wenn in Register R_2 diejenige negative Zahl steht, die den größtmöglichen absoluten Betrag aufweist. Da es bei der Komplementierung dieser Zahl einen Überlauf gibt, ist es gerechtfertigt, wenn dieser auch durch den Bedingungsschlüssel angezeigt wird.

c)

Umladen positiv — Load Positive	10	LPR	RR

$|(R_2)| \rightarrow (R_1)$

Der absolute Wert der im zweiten Register, also in R_2, gespeicherten Zahl wird ins erste Register geladen. Der Bedingungsschlüssel wird auf 0, 2 oder 3 gesetzt. Er kann selbstverständlich niemals auf 1 gesetzt werden, da der Inhalt von R_1 nie negativ sein kann.

d)

Umladen negativ — Load Negative	11	LNR	RR

$-|(R_2)| \rightarrow (R_1)$

Es wird das Zweierkomplement des Absolutwertes der im zweiten Register, also in R_2, stehenden Zahl in das erste Register geladen. Der Bedingungsschlüssel wird auf 0 oder auf 1 gesetzt, je nachdem, ob die ins Register R_1 geladenen Zahl gleich 0 oder negativ ist.

Für die eben besprochenen vier Instruktionen LTR, LCR, LPR und LNR existiert keine äquivalente Instruktion im RX-Format. Für diese vier Instruktionen sollen nunmehr einige Beispiele gegeben werden:

Inhalte vor Ausführung der Instruktionen:

Register 0	0000 0000
Register 1	0000 0001
Register 2	FFFF FFFE
Register 3	8000 0000
Register 4	8000 0001
Register 5	0000 0006
Register 6	0000 0006

1. Instruktion: LTR 12 2 2
Nach dieser Instruktion ist der Inhalt von Register 2 unverändert, der Bedingungsschlüssel ist auf 1 gesetzt, da der Registerinhalt negativ ist.

2. Instruktion: LCR 13 6 2
Nach dieser Instruktion enthält Register 6 den Wert 0000 0002 und der Bedingungsschlüssel ist auf 2 gesetzt, da der Inhalt positiv ist.

3. Instruktion: LPR 10 6 0
Nach dieser Instruktion enthält Register 6 den Wert 0000 0000 und der Bedingungsschlüssel ist auf 0 gesetzt, der der Inhalt gleich 0 ist.

4. Instruktion: LNR 11 6 4
Nach dieser Instruktion enthält Register 6 den Wert 7FFF FFFF und der Bedingungsschlüssel ist auf 2 gesetzt, da der Inhalt positiv ist.

5. Instruktion: LNR 11 6 3
Nach dieser Instruktion enthält Register 6 den Wert 8000 0000 und der Bedingungsschlüssel ist auf 3 gesetzt, wodurch ein Überlauf angezeigt wird. Tatsächlich stellt 8000 0000 die negative Zahl mit dem größtmöglichen absoluten Betrag dar, so daß es bei der Komplementbildung zu einem Überlauf kommt.

6. Instruktion: CR 19 2 1
Die Inhalte der Register 1 und 2 bleiben unverändert, der Bedingungsschlüssel wird auf 1 gesetzt, da $(R_1) > (R_2)$ ist.

Die Instruktion „Verzweigen bei Bedingung" ist auch im RR-Format verfügbar:

Verzweigen bei Bedingung Register — Branch on Condition Register	07	BCR	RR

07	M_1	R_2

Diese Instruktion wirkt genau so wie die bereits besprochene Instruktion BC (echter Operationsschlüssel: 47). Erfolgt die Verzweigung, so wird die Verzweigungsadresse von den letzten 24 Bitstellen des allgemeinen Registers R_2 genommen. Ausnahme: Ist R_2 zu 0 angegeben, so erfolgt grundsätzlich keine Verzweigung, gleichgültig, welches Ergebnis die Prüfung des Bedingungsschlüssels ergab. Damit steht uns in einer anderen Form eine weitere Nop-Instruktion zur Verfügung.

Beispiele:

Vor Ausführung der Instruktionen stehe der Bedingungsschlüssel auf 1 und das Register 9 enthalte den Wert 7F00 4108

1. Instruktion: BCR 07 4 9
Nach Ausführung dieser Instruktion erfolgt eine Verzweigung; die nächste auszuführende Instruktion wird vom Speicherplatz 004 108 geholt.

2. Instruktion: BCR 07 F 0
Es erfolgt keine Verzweigung, es wird die nächstfolgende Instruktion ausgeführt (Nop-Instruktion).

An dieser Stelle wollen wir nunmehr noch zwei weitere Verzweigungsinstruktionen kurz besprechen:

a)

Verzweigen und Laden Folgeadresse — Branch and Link	45	BAL	RX

b)

Verzweigen und Laden Folgeadresse (Reg.) — Branch and Link	05	BALR	RR

Wenn diese Instruktionen ausgeführt werden, so wird die Adresse der darauffolgenden Instruktion (Folgeadresse) im allgemeinen Register R_1 gespeichert; es erfolgt außerdem eine Verzweigung zu der Adresse, die durch E_2 angezeigt wird (Summation von X_2, B_2, und D_2) bzw. die im Register R_2 steht (beim RR-Format). Im RR-Format erfolgt keine Verzweigung, wenn für R_2 Null angegeben wird. Die Folgeadresse wird in den letzten 24 Bitstellen des Registers R_1 gespeichert, im ersten Byte dieses Registers erfolgt die Speicherung anderer für die Verzweigung wichtiger Informationen. Im einzelnen werden wir diese Instruktionen später im Detail besprechen (Abschnitt 7.8.).

Eine höchst wichtige Verwendung findet die Instruktion BALR, um ein Basisregister einzurichten. Hierzu wird die Adresse der nächsten Instruktion in ein beliebiges allgemeines Register ungleich des Registers 0 geladen, um dieses dann als Basisregister einsetzen zu können. Die einem solchen Vorhaben entsprechende Codierung lautet

BALR R_1,0

Die Adresse des Hauptspeicherplatzes, der die um 2 erhöhte Adresse des Hauptspeicherplatzes aufweist, auf dem die BALR-Instruktion selbst beginnt, wird mittels dieser Instruktion ins allgemeine Register R_1 geladen.

Sehr oft bildet eine Instruktion dieses Aussehens die erste Instruktion eines Programmes. Auf alle anderen Adressen innerhalb des Programmes kann man sich dann mit Hilfe von Distanzadressen beziehen, die in Relation zu der durch die BALR-Instruktion geschaffenen Basisadresse gebildet werden müssen. Auf diese Weise ist es möglich, daß ein Programm in ein anderes Hauptspeichergebiet gelegt werden und dennoch weiterhin korrekt ablaufen kann.

4.5. Eingabe und Ausgabe (EA)

Bis hierhin haben wir eine genügende Anzahl Instruktionen aus dem Instruktionsvorrat des Systems IBM /370 behandelt, um einfache Programme schreiben zu können, die jedoch frei von EA-Operationen sein müssen. Wir sind ferner noch nicht damit vertraut, wie ein Programm in den Hauptspeicher des Computers geladen werden kann und wie darüberhinaus die Ablaufsteuerung der ersten Instruktion eines Programmes übergeben werden kann. Nehmen wir deshalb zunächst an, daß es irgendwelche einfachen Wege und Methoden gibt, um diese unerläßlichen Vorkehrungen bewältigen zu können. Später werden wir sehen, daß es von der jeweiligen Installation abhängig ist, wie wir in dieser Hinsicht vorzugehen haben.

Zum gegenwärtigen Zeitpunkt wollen wir alle EA-Operationen nur symbolisch andeuten. Die in diesem Abschnitt behandelten drei Symbolinstruktionen genügen uns vorläufig, um eine Reihe recht nützlicher Programme abfassen zu können.

1. Stanzen von Lochkarten
   Durch die Symbolinstruktion

   PUNCH adresse

   erreichen wir, daß eine 80 spaltige Lochkarte gestanzt wird. Die auf sie übertragenen Informationen beginnen bei der Hauptspeicheradresse adresse und reichen bis zur Hauptspeicheradresse adresse+79. Die im Hauptspeicher stehenden Daten bleiben dabei unverändert bestehen. Nach dem Stanzen der Lochkarte geht das Programm zur unmittelbar anschließenden Folgeinstruktion über.

2. Lesen von Lochkarten
   Durch die Symbolinstruktion

   READ adresse,deadr

   erreichen wir, daß eine 80 spaltige Lochkarte gelesen wird. Ihr Inhalt ersetzt die bisher auf den Hauptspeicherplätzen von adresse bis adresse+79 einschließlich vorhandenen Informationen. Nach dem Lesen der Lochkarte wird die darauffolgende Instruktion ausgeführt. Stößt die Programmablaufsteuerung auf eine Symbolinstruktion READ und ist die letzte in der Zuführung eines Kartenlesegerätes liegende

Karte bereits gelesen, dann bleibt der Inhalt der Hauptspeicherplätze von adresse bis adresse+79 einschließlich ungeändert und die nächste auszuführende Instruktion wird auf dem Hauptspeicherplatz mit der Adresse deadr[1] gesucht; dort muß also eine Instruktion beginnen.

3. Drucken einer Zeile
Durch die Symbolinstruktion

PRINT adresse

erreichen wir, daß mittels eines Zeilendruckers eine Zeile mit insgesamt 120 Druckstellen gedruckt wird. Auf ihr erscheinen die Daten, die im Hauptspeicher auf den Plätzen gespeichert sind, die bei der Adresse adresse beginnen und bis einschließlich adresse+119 reichen. Die im Hauptspeicher auf den genannten Stellen stehenden Daten bleiben dabei unverändert erhalten. Nach dem Drucken der Zeile kommt es zur Ausführung der unmittelbar an diese Symbolinstruktion anschließenden Instruktion. Wir setzen weiterhin voraus, daß zu Beginn des Programmablaufes das Druckpapier auf den Anfang einer Seite eingestellt ist und daß gerade sechzig Zeilen, jeweils im Abstand einer Zeile aufeinander folgend, auf eine Zeile passen.

Neben diesen drei, stellvertretend für die eigentlichen EA-Instruktionen erscheinenden Symbolinstruktionen wollen wir noch eine vierte Symbolinstruktion einführen, mit deren Hilfe wir das Ende eines Programmes festlegen können. Die Symbolinstruktion

EOJ

hat auf die letzte Instruktion zu folgen, die in einem Programm zur Ausführung gelangen soll. Sie zeigt das Arbeitsende [2] des betreffenden Programmes an.

Wir nehmen weiterhin an, daß jede der soeben genannten vier Symbolinstruktionen gerade 16 Stellen des Hauptspeichers belegt. Außerdem wollen wir strikt darauf achten, daß wir die allgemeinen Register 13 und 14 nicht mehr benutzen. Nach Übereinkunft sollte nämlich bei den Systemen IBM /370 die Verwendung dieser Register für bestimmte spezielle Zwecke reserviert bleiben, über die wir uns später noch im Detail unterhalten werden. Doch vorerst schadet es wohl nichts, wenn wir lernen, möglichst frühzeitig an die Existenz derartiger Konventionen zu denken.

Damit wir uns an unsere soeben eingeführten Absprachen auch gewöhnen, wollen wir ein kleines Programm schreiben, das folgende Aufgabenstellung löst. Es soll ein Kartenpaket eingelesen werden, dessen einzelne Karten auf Spalte 4 den Wert von x und auf Spalte 8 den Wert von y enthalten, beide Werte jeweils im Wertebereich [0, 9] liegend. Alle anderen Kartenspalten enthalten nur Leerzeichen. Es soll

$$z = |x-y|$$

errechnet werden, d.h. der absolute Betrag der Differenz von x und y. Nach Durchführung dieser arithmetischen Operation soll eine Zeile gedruckt werden, in der auf Druckstelle 4 der Wert von x, auf Druckstelle 8 der Wert von y und auf Druckstelle 12 der Wert von z erscheinen. Der Rest der zu druckenden Zeile soll frei bleiben. Die Lösung dieser Aufgabe finden wir in Abb. 4.5.

1) deadr soll symbolisch für Datenendeadresse stehen.
2) Im angloamerikanischen Sprachraum als „End of Job", abgekürzt EOJ bezeichnet.

Hauptspeicherplatz bzw. Hauptspeicherplätze	Inhalt (Instruktionen bzw. Daten)	Bemerkungen
Ø2Ø ØØØ	Ø5 B Ø	Laden des Basisregisters 11 mit der nächstfolgenden Adresse, hier 020 002
Ø2Ø ØØ2	READ Ø2Ø1ØØ,Ø2ØØ5Ø	Lesen einer Lochkarte
Ø2Ø Ø12	58 3 Ø B ØFE	Laden des eingelesenen x-Wertes ins Register 3, das somit den Inhalt 404040Fx (sedezimal) aufweist
Ø2Ø Ø16	1B 2 2	Setzen des Registers 2 auf 0 durch Subtraktion
Ø2Ø Ø18	5D 2 Ø B 176	Durch Division wird der Inhalt des Registers 2 auf 0000000x (sedezimal) gebracht
Ø2Ø Ø1C	58 5 Ø B 1Ø2	Laden des eingelesenen y-Wertes in Register 5, das somit den Inhalt 404040Fy (sedezimal) aufweist
Ø2Ø Ø2Ø	1B 4 4	Setzen des Registers 4 auf 0 durch Subtraktion
Ø2Ø Ø22	5D 4 Ø B 176	Durch Division wird der Inhalt des Registers 4 auf 0000000y (sedezimal) gebracht
Ø2Ø Ø26	1B 2 4	Differenz x-y im Register 2
Ø2Ø Ø28	1Ø 2 2	Absolutbetrag x-y im Register 2
Ø2Ø Ø2A	5A 2 Ø B 17A	Durch Addition der auf 020 17C gespeicherten Konstanten wird der Inhalt des Registers 2 auf 404040Fz gebracht
Ø2Ø Ø2E	5Ø 2 Ø B 1Ø6	Speicherung des Inhalts von Register 2 auf die Druckstellen 9 bis 12
Ø2Ø Ø32	PRINT Ø2Ø1ØØ	Drucken einer Zeile
Ø2Ø Ø42	47 F Ø B ØØØ	Unbedingte Verzweigung zum Programmanfang d.h. zum Speicherplatz mit der Adresse 020 002
Ø2Ø Ø46	unbenutzt	——
Ø2Ø Ø5Ø	EOJ	Programmende, angesprungen durch Datenende bei READ
Ø2Ø 1ØØ bis Ø2Ø 14F	Eingabegebiet für die Lochkarten	——
Ø2Ø 15Ø bis Ø2Ø 177	Leerzeichen, d.h. 4Ø (sedezimal) auf jeder Stelle	Erweiterung des Eingabegebietes um 40 Stellen, um ein Druckgebiet von insgesamt 120 Stellen zu erhalten
Ø2Ø 178 bis Ø2Ø 17B	ØØØØØØ1Ø	Konstante 16 (dezimal) für die beiden Divisionen
Ø2Ø 17C bis Ø2Ø 17F	4Ø4Ø4ØFØ	Konstante für die Additionsinstruktion auf 020 02A

Abb. 4.5. Beispielprogramm

Bei der Betrachtung des Beispielprogrammes (Abb. 4.5.) sollten wir uns immer vor Augen halten, daß wir als Basisregister das Register 11 (sedezimal B) ausgewählt haben und daß dieses Register infolge der Instruktion BALR (siehe Abschnitt 4.4.) mit der Basisadresse 020002 geladen wird. Alle Bezugnahmen auf Adressen innerhalb dieses Programmes erfolgen deshalb mit Distanzadressen, die gleich der Differenz zwischen der effektiven tatsächlichen Adresse und der Basisadresse sind.

Aufgabengruppe 9

1) Gegeben sind die folgenden Inhalte:

001 000 bis 001 003	0000 0005
001 004 bis 001 007	FFFF F38A
001 008 bis 001 00B	0000 1764
Register 0	0000 4000
Register 1	0000 1000
Register 2	0000 0008

Es sei ferner vorausgesetzt, daß der Bedingungsschlüssel auf 3 steht. Welchen Wert weist der Bedingungsschlüssel auf, wenn die nachfolgenden Instruktionen ausgeführt werden?

a)	LTR	12 2 2
b)	CR	19 2 2
c)	C	59 2 1 0 008
d)	A	5A 0 1 0 004
e)	LCR	13 2 0
f)	LPR	10 2 0
g)	S	5B 2 1 0 004
h)	L	58 2 1 0 004
i)	SR	1B 0 0
j)	BC	47 4 1 0 020

2) Gegeben sind die folgenden Konstanten:

001 00C bis 001 00F	7000 6320
001 010 bis 001 013	8012 0000
Register 1	0000 1000
Register 2	0000 0008
Register 3	FFF3 0004

Ermittle den Speicherplatz der nächsten auszuführenden Instruktion, wenn die nachstehende Instruktionsfolge abgelaufen ist!

a)	002 000	LNR	11 33
		BC	47 4 1 0 020
b)	002 000	AR	1A 1 1
		BC	47 1 1 0 020
c)	002 000	L	58 3 1 0 00C
		AR	1A 3 3
		BC	47 1 1 0 100
d)	002 000	L	58 3 1 0 010
		AR	1A 3 3
		BC	47 1 1 0 100

3) Ändere das Beispielprogramm vom Abschnitt 4.5. (Abb. 4.5.) wie folgt ab: Zu Beginn ist eine Überschriftszeile zu drucken, deren erste zwölf Druckstellen den Inhalt

bbbXbbbYbbbZ

aufweisen (b bedeutet dabei das Leerzeichen). Die Druckstellen 13 bis 120 bleiben leer.

4) Das Beispielprogramm vom Abschnitt 4.5. (Abb. 4.5.) ist wie folgt abzuändern: Die auszugebenden Werte sind in der Reihenfolge y, x und z an Stelle von x, y und z zu drucken.

5) Modifiziere das Beispielprogramm vom Abschnitt 4.5. (Abb. 4.5.) wie folgt: Es ist z=x-y an Stelle des absoluten Betrages von z zu errechnen. Ergibt sich dabei ein negativer Wert für z, so ist auf der Druckstelle 11 zusätzlich ein Minuszeichen zu drucken.

6) Gegeben sind die Inhalte der folgenden Hauptspeicherplätze:

Adresse(n)	Inhalt (Daten)
2Ø19ØØ bis 2Ø19ØF	5ØE3C8C9 E24ØC9E2 4ØC14ØD7 D9D6C2D3
2Ø191Ø bis 2Ø191F	C5D45A4Ø 4Ø4Ø4Ø4Ø 4Ø4Ø4Ø4Ø 4Ø4Ø4Ø4Ø
2Ø192Ø bis 2Ø1977	Leerzeichen (sedezimal 4Ø) auf jeder Stelle
2Ø1978 bis 2Ø1987	ØØØØ4Ø4Ø 4ØØØ4Ø4Ø 1Ø2247B2 ØØØØØØØØ

Das auszuführende Programm weist die nachstehende Instruktionsfolge auf, die vom Hauptspeicherplatz mit der Adresse 201800 an beginnt:

```
Ø5 B Ø
58 2 Ø B ØFE
5B 2 Ø B 176
5Ø 2 Ø B ØFE
58 3 Ø B 1Ø2
5B 3 Ø B 17A
5B 2 Ø B 17E
5Ø 2 Ø B 1Ø6
58 4 Ø B 1ØA
5B 4 Ø B 12A
5Ø 4 Ø B 1ØA
58 5 Ø B 1ØE
5B 5 Ø B 12A
5A 5 Ø B 176
5Ø 5 Ø B 12A
PRINT 2Ø19ØØ
EOJ
```

Was wird auf der Zeile gedruckt?

7) Kann man durch Änderung des Programmes von Aufg. 6) erreichen, daß die Druckausgabe mit den passenden Groß- und Kleinbuchstaben erfolgt?
Änderungsvorgabe: a) Hinzufügung einer Instruktion
b) Änderung einer oberhalb der Adresse 201978 gespeicherten Konstanten

4.6. Instruktionen im SS-Format

Alle bisher behandelten Instruktionen sprachen Wörter an. In diesem Abschnitt wollen wir uns mit Instruktionen befassen, die Operationen mit Feldern unterschiedlicher Länge bewirken. Diese Instruktionen arbeiten im SS-Format (Speicher zu Speicher), das so aufgebaut ist, wie es die Abb. 4.6. zeigt:

Byte	0	1	2	3	4	5
Inhalt des Bytes	Operations-schlüssel	L	B_1	D_1	B_2	D_2

Abb. 4.6. SS-Format von Instruktionen

Hierbei bedeutet L den sogenannten Längenschlüssel. Dieser drückt aus, wieviele Bytes in die betreffende Operation verwickelt werden. L kann Werte zwischen 0 und 255 (einschließlich) annehmen und ist immer um 1 kleiner als die Anzahl der von der Operation betroffenen Bytes, d.h. 1 bis 256 Bytes können von Instruktionen dieses Formats angesprochen werden. In allen Fällen werden die Daten vom zweiten Feld (adressiert über B_2 und D_2) ins erste Feld (adressiert über B_1 und D_1) übertragen. Das zweite Feld bleibt immer dann unverändert, wenn nicht das erste Feld das zweite Feld überlappt.

Nachstehend werden der Reihe nach die einzelnen Instruktionen kurz besprochen.

1) | Übertragen Zeichen — Move Characters | D2 | MVC | SS |

Der Inhalt des durch B_2 und D_2 angesprochenen Feldes wird in das durch B_1 und D_1 angesprochene Feld übertragen. Die Daten werden seriell Byte auf Byte übertragen, beginnend an der linken Begrenzung des Feldes. Die aus den Basisadressen und den Distanzadressen zusammengesetzten Feldadressen sind deshalb gleich den Adressen der am weitesten links stehenden Bytes der Felder.

Beispiele:

Vorher:

Register 1	0000 4000
Speicherplätze 004 000	00 01 02 03 04 05
bis 004 008	06 77 88
Speicherplätze 004 010	EE DD CC BB AA 99
bis 004 016	34

1. Instruktion: MVC D2 06 1 000 1 010

Nach dieser Instruktion:

Speicherplätze 004 000	EE DD CC BB AA 99
bis 004 008	34 77 88
Speicherplätze 004 010	unverändert
bis 004 016	

2. Instruktion: MVC D2 04 1 012 1 011

Nach dieser Instruktion:

Speicherplätze 004 010	EE DD DD DD DD DD DD
bis 004 016	

2)

Übertragen Ziffern — Move Numerics	D1	MVN	SS

Der numerische Anteil jedes Bytes des zweiten Operanden ersetzt den entsprechenden Anteil jedes Bytes des ersten Operanden. Der Zonenteil der Bytes verbleibt unverändert. Diese Instruktion ist also äquivalent der Instruktion MVC mit der Ausnahme, daß nur die rechten Halbbytes übertragen werden.

3)

Übertragen Zonen — Move Zones	D3	MVZ	SS

Der Zonenteil jedes Bytes des zweiten Operanden ersetzt den Zonenteil jedes Bytes des ersten Operanden; die Zifernteile (numerische Anteile) der Bytes bleiben unverändert. Diese Instruktion ist also äquivalent der Instruktion MVC mit der Ausnahme, daß nur die linken Halbbytes übertragen werden.

Für die Instruktionen MVN und MVZ folgen jetzt Beispiele:

Vorher:

Register 1	0000 4000
Speicherplätze 004 000	00 01 02 03 04 05
bis 004 008	06 77 88
Speicherplätze 004 010	EE DD CC FB AA 99 34
bis 004 016	

1. Instruktion: MVN D1 03 1 005 1 010

Nach dieser Instruktion:

Speicherplätze 004 000	00 01 02 03 04 0E
bis 004 008	0D 7C 8B
Speicherplätze 004 010	unverändert
bis 004 016	

2. Instruktion: MVZ D3 02 1 000 1 012

Nach dieser Instruktion:

Speicherplätze 004 000	C0 F1 A2
bis 004 002	
Alle anderen Speicherpl.	unverändert

Die Instruktionen MVN und MVZ sind in erster Linie nützlich, wenn es gilt, Informationen aufzubereiten, die von Karten gelesen werden oder die in Karten zu stanzen sind. Sehr

häufig findet man Zonenlochungen auf Kartenspalten, die zu numerischen Feldern gehören, jedoch in keinerlei Beziehung zu diesen stehen (Vorzeichen usw.). Derartige Informationen können von den anderen durch diese Instruktionen abgesondert werden, bevor mit der sonstigen Verarbeitung begonnen wird.

4.7. Gepackte Daten (Daten in gepackter Form)

Da ein Byte zwei Dezimalziffern aufnehmen kann, können wir bei großen Datenmengen viel Speicherplatz sparen, wenn wir zwei Dezimalziffern in ein einziges Byte hineinstellen. Dieses Vorgehen bezeichnet man als Packen. Für das Packen und Entpacken stehen uns zwei Instruktionen zur Verfügung. Beide besitzen das SS-Format, doch unterscheiden sie sich geringfügig von den Instruktionen, die wir im vorigen Abschnitt 4.6. behandelt haben.

Byte	0	1		2	3	4	5
Inhalt des Bytes	Operations-schlüssel	L_1	L_2	B_1	D_1	B_2	D_2

Abb. 4.7. SS-Format beim Packen und Entpacken

Bei diesen Instruktionen werden die Felder im allgemeinen nicht die gleiche Länge besitzen. Deshalb muß jedes Feld durch seinen eigenen Längenschlüssel gekennzeichnet werden. Zur Angabe dieser Längen wird nur eine einzige Sedezimalziffer verwendet, so daß die Längenschlüssel nur Werte zwischen 0 und 15 annehmen können, was Feldlängen von 1 bis 16 entspricht.

1)

Packen — Pack	F2	PACK	SS

Das durch den zweiten Operanden angesprochene Feld wird vom ungepackten Zustand in den gepackten Zustand umgewandelt und in das durch den ersten Operanden adressierte Feld gespeichert. (Felder im ungepackten Zustand wollen wir der Einfachheit halber auch als Felder mit Zonenteile bezeichnen). Ist das erste Feld nicht groß genug, um das gesamte Resultat aufnehmen zu können, so werden nur die Dezimalziffern niedriger Ordnung (die rechten also) übernommen, während die Dezimalziffern höherer Ordnung (die linksstehenden also) verloren gehen. Ist dagegen das durch den ersten Operanden adressierte Feld zu groß, so wird es nach links mit Nullen aufgefüllt. In der folgenden Abbildung 4.8. wird dargestellt, welches Ergebnis beim Packen eines sechs Bytes langen Feldes in ein vier Byte langes Feld erhalten wird.

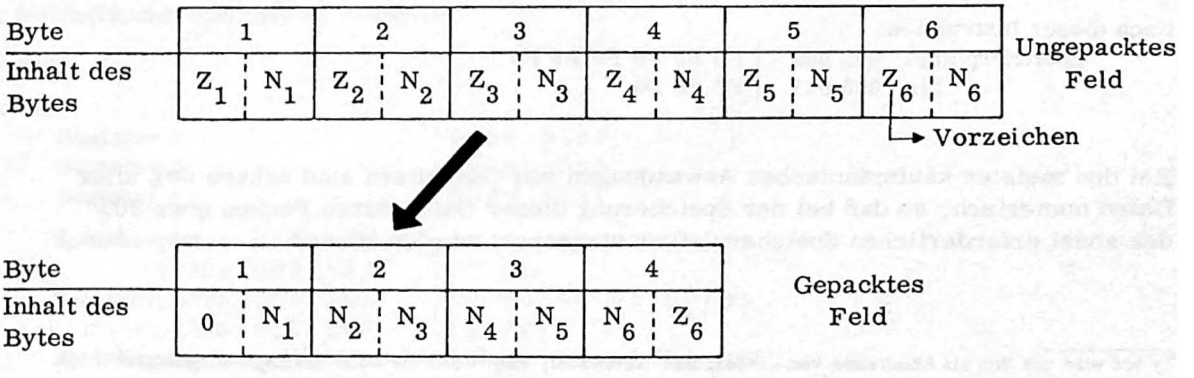


Abb. 4.8. Beispiel für das Packen eines Feldes

Links wird, wie man sieht, eine Null eingeschleust, damit das überzählige Halbbyte ebenfalls ausgefüllt ist. Für das Feld mit den Zonenteilen wird angenommen, daß der Zonenteil des letzten Bytes (der Einerstelle also) ein Vorzeichen enthält, entweder + oder -. Dieses Vorzeichen wird unverändert in das letzte Halbbyte des gepackten Feldes übertragen. Die anderen Zonenteile des ungepackten Feldes bleiben unbeachtet. Beginnend am rechten Ende des ungepackten Feldes werden anschließend alle Ziffernteile in fortlaufende Halbbytes des gepackten Feldes unverändert übertragen. Somit ist die Anzahl der Dezimalziffern in einem gepackten Feld stets ungerade; sie kann zwischen 1 und 31 groß sein.

2)

Entpacken — Unpack	F3	UNPK	SS

Durch diese Instruktion wird ein durch den zweiten Operanden angesprochenes gepacktes Feld in ein ungepacktes Feld umgewandelt, das durch den ersten Operanden angesprochen wird. Das Vorzeichen des gepackten Feldes wird in den Zonenteil des letzten Bytes des ungepackten Feldes übertragen; die Zonenteile der restlichen Bytes des ungepackten Feldes werden auf F(sed) [1] gesetzt, da dies der normale Zonenteil bei Ziffern ist, die im EBCDI-Code verschlüsselt sind. Ist das ungepackte Feld zu kurz angegeben, so gehen die Ziffern höherer Ordnung des gepackten Feldes bei der Umwandlung verloren. Ist das ungepackte Feld dagegen zu lang angegeben, so werden nach links Nullen eingefügt.

Für die beiden Instruktionen PACK und UNPK sollen nun noch einige Beispiele angeführt werden:

Vorher:

Register 1		0000 5000
Speicherplätze	005 000	F3 F4 F2 F3 F0 F5
bis	005 008	F6 F7 D8
Speicherplätze	005 009	09 0A 0B 0C 43 7E
bis	005 011	0F 10 11
Speicherplätze	005 012	56 78 9D
bis	005 014	

1. Instruktion: PACK F2 8 8 1 009 1 000

Nach dieser Instruktion:

Speicherplätze	005 009	00 00 00 00 34 23
bis	005 011	05 67 8D

2. Instruktion: UNPK F3 8 2 1 009 1 012

Nach dieser Instruktion:

Speicherplätze	005 009	F0 F0 F0 F0 F5 F6
bis	005 011	F7 F8 D9

Bei den meisten kaufmännischen Anwendungen von Computern sind nahezu 60% aller Daten numerisch, so daß bei der Speicherung dieser Daten durch Packen etwa 30% des sonst erforderlichen Speicherplatzes eingespart werden können.

1) sed wird von uns als Abkürzung von „sedezimal" verwendet; damit soll die Basis des zugrundegelegten Zahlensystems angegeben werden.

4.8. Dezimal-Dual-Konvertierung

Zwei Instruktionen der IBM /370 erleichtern die Konvertierung von gepackten Dezimalzahlen in Dualzahlen und umgekehrt. Die gepackte Dezimalzahl muß daher in einem Doppelwort des Hauptspeichers gespeichert sein (das bedeutet, daß sie acht Bytes umfassen und daß das linke Byte eine durch 8 teilbare Adresse besitzen muß); für die Dualzahl dagegen muß ein allgemeines Register vorgesehen werden. Die zu konvertierenden Zahlen werden als ganze Dezimalzahlen bzw. als ganze Dualzahlen angesehen; negative ganze Dualzahlen müssen im Zweierkomplement vorliegen. Negative Dezimalzahlen müssen nach der Einerstelle folgende Minuszeichenverschlüsselung aufweisen (im rechten Halbbyte natürlich): D oder B.

1)

Konvertieren ins Dualsystem — Convert to Binary	4F	CVB	RX

Die in einem Doppelwort stehende gepackte Dezimalzahl, die durch den zweiten Operanden angesprochen wird, wird in eine ganze Dualzahl konvertiert und im allgemeinen Register R_1 gespeichert. Steht in der Vorzeichenposition der gepackten Dezimalzahl ein B (sed) oder ein D(sed), so wird die Dualzahl zusätzlich in ihr Zweierkomplement überführt. Steht in der Vorzeichenposition der gepackten Dezimalzahl dagegen ein A (sed), ein C (sed), ein E (sed) oder ein F (sed), so ergibt sich eine positive ganze Dualzahl.

2)

Konvertieren ins Dezimalsystem — Convert to Decimal	4E	CVD	RX

Die im Register R_1 stehende ganze Dualzahl wird in eine gepackte ganze Dezimalzahl konvertiert, die in dem Doppelwort gespeichert wird, das durch E_2 adressiert ist. Eine negative Dualzahl erzeugt auf der Vorzeichenposition der Dezimalzahl ein sedezimales D (EBCDI-Code); eine positive Dualzahl erzeugt dagegen ein sedezimales C (EBCDI-Code).

Der Wertebereich, in dem die zu konvertierenden Zahlen liegen dürfen, ist nur durch den Wertebereich begrenzt, der für die in einem allgemeinen Register stehenden Zahlen gilt. Somit müssen die Zahlen größer oder gleich -2 147 483 648 bzw. kleiner oder gleich +2 147 483 647 sein. Liegt bei der Dezimal-Dual-Konvertierung die gepackte Dezimalzahl außerhalb dieses Wertebereiches, so wird die Konvertierung durchgeführt, doch nur die 32 rechten Dualziffern der entstandenen Dualzahl werden im allgemeinen Register R_1 gespeichert.

Es ist erlaubt, daß in die Operation ein Doppelwort einbezogen wird, das sich nicht auf der Doppelwortgrenze befindet. In diesem Fall erhöht sich die Ausführungszeit der betreffenden Instruktion erheblich.

Beispiele:

Vorher:

Register 1	0000 5100
Register 2	0000 0002
Register 3	FFFF 1387
Speicherplätze 005 100	00 00 00 00 00 00
bis 005 107	12 3D
Speicherplätze 005 108	00 00 00 00 00 00
bis 005 10F	12 3C
Speicherplätze 005 110	13 34 67 7D 8B A5
bis 005 117	C2 00

```
1. Instruktion: CVB          4F  2  0  1  000

Nach dieser Instruktion:
      Register 2                    | FFFF FF85

2. Instruktion: CVB          4F  2  0  1  008

Nach dieser Instruktion:
      Register 2                    | 0000 007B

3. Instruktion: CVD          4E  3  0  1  008

Nach dieser Instruktion:
      Speicherplätze 005 108        | 00 00 00 00 00 60
                bis  005 10F        | 53 7D

4. Instruktion: CVD          4E  1  0  1  008

Nach dieser Instruktion:
      Speicherplätze 005 108        | 00 00 00 00 00 20
                bis  005 10F        | 73 6C
```

Verwenden wir sowohl die Instruktion PACK als auch die Instruktion CVB, so können wir leicht eine Zahl, die in ungepackter Form eingelesen wurde, in eine ganze Dualzahl konvertieren. Nach der Verarbeitung können wir das Ergebnis durch Einsatz der Instruktionen CVD und UNPK in eine ungepackte Dezimalzahl konvertieren und diese dann ausgeben. Betrachten wir dazu das folgende kleine Programm, das zwei sechsstellige ungepackte Dezimalzahlen auf den Speicherplätzen 005 200 und 005 206 addieren und die Summe als ungepackte Dezimalzahl auf den Speicherplatz 005 20C speichern soll.

Inhalt des Registers 1: 0000 5200

Instruktionen:

Mnemonischer Operationsschlüssel	Maschinencode
PACK	F2 7 5 1 018 1 000
CVB	4F 2 0 1 018
PACK	F2 7 5 1 018 1 006
CVB	4F 3 0 1 018
AR	1A 2 3
CVD	4E 2 0 1 018
UNPK	F3 5 7 1 00C 1 018

Abb. 4.9. Beispielprogramm

In diesem Beispielprogramm haben wir uns ein Arbeitsgebiet angelegt, in dem wir die gepackten Dezimalzahlen bilden. Das Arbeitsgebiet umfaßt die Bytes mit den Adressen von 005 218 bis 005 21F. In den Registern 2 und 3 sind die Dualzahlen festgehalten worden. Es sollte außerdem vermerkt werden, daß nur eine der sieben Instruktionen eine arithmetische Operation bewirkt; die restlichen Instruktionen befassen

sich mit den erforderlichen Datentypumwandlungen. Bei kommerziellen Anwendungen von Computern ist es oft typisch, daß die Zahl der Rechenoperationen von den anderen Operationen bei weitem übertroffen wird.

4.9. Dezimalarithmetik

Im Abschnitt 4.8. setzten wir uns am Schluß mit einem Beispiel (siehe Abb. 4.9.) auseinander, bei dem die Zahl der Rechenoperationen verschwindend klein gegenüber der Zahl der sonstigen Operationen war. Um solche Anwendungen zu erleichtern, sind eine Reihe von Instruktionen verfügbar, mit denen unmittelbar die Dezimalarithmetik betrieben werden kann. Diese Instruktionen liegen im SS-Format mit zwei Längenangaben vor (Abb. 4.10.).

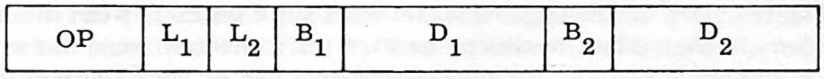


Abb. 4.10. SS-Format der Instruktionen für die Dezimalarithmetik

Nachstehend werden diese Instruktionen kurz besprochen.

1) Addieren dezimal — Add Decimal | FA | AP | SS

Der Inhalt des durch den zweiten Operanden adressierten Feldes wird zum Inhalt des durch den ersten Operanden adressierten Feldes addiert; die Summe ersetzt den Inhalt des ersten Feldes. Die Felder des ersten und des zweiten Operanden dürfen sich nur dann überlappen, wenn die rechten Stellen (Einerstellen) der Felder zusammenfallen. Das Vorzeichen der Summe ergibt sich gemäß der algebraischen Regeln; eine Summe, die gleich Null ist, besitzt das positive Vorzeichen. Der Bedingungsschlüssel wird auf 0, 1 oder 2 gesetzt, wenn die Summe gleich Null, kleiner als Null oder größer als Null ist. Ist die Summe jedoch zu groß, um vom Ergebnisfeld aufgenommen werden zu können, so wird der Bedingungsschlüssel auf 3 gesetzt. Auf diese Weise wird also ein Überlauf angezeigt. Beim Überlauf gehen die höherwertigen Ziffern des Resultates verloren.

2) Subtrahieren dezimal — Subtract Decimal | FB | SP | SS

Der zweite Operand wird vom ersten Operanden subtrahiert, die Differenz ersetzt den ersten Operanden. Der Bedingungsschlüssel wird wie bei der Instruktion AP gesetzt. Diese Instruktion läuft genau so wie die Instruktion AP ab, nur daß der Computer intern zuvor das Vorzeichen des zweiten Operanden umkehrt. Fallen die rechten Stellen zweier sich überlappender Felder zusammen, so kann diese Instruktion dazu benutzt werden, um ein Feld oder einen Teil desselben auf Null zu setzen.

3) Löschen und addieren — Zero and Add | F8 | ZAP | SS

Der zweite Operand wird in das Feld des ersten Operanden gestellt. Der Bedingungsschlüssel wird genau so wie bei der Instruktion AP gesetzt. Ist das Feld des ersten Ope-

randen zu groß, so wird es nach links mit Nullen aufgefüllt, ist es zu kurz, so gehen die führenden Ziffern verloren. Geht dabei eine Ziffer verloren, die ungleich Null ist, so wird der Bedingungsschlüssel auf 3 gesetzt, auf diese Weise einen Überlauf anzeigend. Die beiden Operandenfelder dürfen sich nur dann überlappen, wenn die rechten Stellen (Einerstellen) der Felder zusammenfallen. Bei Ausführung dieser Instruktion wird das Feld des ersten Operanden zunächst auf Null gesetzt und anschließend wird der Inhalt des Feldes des zweiten Operanden auf das erste Feld addiert.

4)

Vergleichen dezimal — Compare Decimal	F9	CP	SS

Diese Instruktion ist identisch der Instruktion SP, nur daß das Resultat des Subtrahierens verloren geht und nicht gespeichert wird. Beide Operandenfelder verbleiben also ungeändert erhalten. Der Bedingungsschlüssel wird auf 0 gesetzt, wenn die beiden Operandenfelder den gleichen Inhalt besitzen; er wird auf 1 gesetzt, wenn das erste Operandenfeld einen kleineren Inhalt als das zweite aufweist, und er wird schließlich auf 2 gesetzt, wenn das erste Operandenfeld einen größeren Inhalt als das zweite besitzt. Da kein Resultat gespeichert wird, kann auch kein Überlauf eintreten. Eine Null mit positivem Vorzeichen wird von dieser Instruktion als gleichwertig mit einer Null mit negativem Vorzeichen betrachtet.

5)

Multiplizieren dezimal — Multiply Decimal	FC	MP	SS

Das Produkt aus Multiplikator (zweiter Operand) und Multiplikand (erster Operand) wird im Feld des ersten Operanden gespeichert. Der Multiplikator ist auf 15 Ziffernstellen plus einer Vorzeichenstelle (entspricht acht Bytes) beschränkt; außerdem muß er stets kürzer sein als der Multiplikand. Da die Stellenzahl des Produktes gleich der Summe der Stellenzahlen der beiden Faktoren ist, muß das Multiplikandenfeld mindestens so viele führende Nullen enthalten wie der Multiplikator Stellen enthält. Dadurch ist gesichert, daß kein Überlauf stattfinden kann. Die beiden Operandenfelder können sich nur dann überlappen, wenn ihre rechten Bytes zusammenfallen. Der Bedingungsschlüssel wird durch diese Operation nicht geändert. Um das erste Operandenfeld, wenn notwendig, zu erweitern, kann man vorteilhaft die Instruktion ZAP einsetzen. Die maximale Länge des Produktes kann 31 Stellen nicht überschreiten.

6)

Divdieren dezimal — Divide Decimal	FD	DP	SS

Der Dividend (erster Operand) wird durch den Divisor (zweiter Operand) dividiert; Quotient und Rest werden im Dividendenfeld gespeichert. Der Quotient wird dabei linksbündig in das Feld des ersten Operanden gestellt und weist eine Länge von L_1-L_2 Bytes auf. Der Rest wird im Feld des ersten Operanden im Anschluß an den Quotienten gespeichert; seine Adresse ist also E_1+L_1-L_2, seine Länge beträgt L_2 Bytes. Der Divisor darf nicht mehr als acht Bytes (15 Ziffernstellen und eine Vorzeichenstelle) umfassen. Außerdem muß er weniger Bytes als der Dividend aufweisen. Alle in die Division eingehenden Werte werden als ganze Zahlen angesehen. Das Vorzeichen des Quotienten wird nach den Regeln der Algebra gebildet, der Rest besitzt grundsätzlich das gleiche Vorzeichen wie der Divisor. Diese Feststellung besitzt auch dann Gültigkeit, wenn der

Rest gleich Null ist. Die Operandenfelder dürfen sich nur dann überlappen, wenn ihre rechten Bytes zusammenfallen. Der Bedingungsschlüssel verbleibt ungeändert. Maximal darf der Dividend 31 Ziffernstellen besitzen, der Quotient ist auf 29 Ziffernstellen begrenzt.

Die nachfolgende Instruktion ist keine arithmetische Instruktion, doch soll sie ihres Verwendungszweckes wegen an dieser Stelle diskutiert werden.

7)

Umpacken — Move with Offset	F1	MVO	SS

Der Inhalt des durch den zweiten Operanden angesprochenen Feldes wird um ein halbes Byte (vier Bitstellen) nach links versetzt; in das damit freiwerdende letzte Halbbyte wird das Vorzeichen des ersten Operandenfeldes hineingestellt. Das so gebildete Feld wird in das durch den ersten Operanden adressierte Feld übertragen. Die Felder werden von rechts nach links verarbeitet. Wenn notwendig, wird das zweite Operandenfeld durch Nullen erweitert. Ist das erste Operandenfeld zu kurz, so gehen die führenden Stellen des zweiten Operandenfeldes verloren. Die Felder dürfen sich überlappen. Durch diese Instruktion erreicht man, daß das zweite Operandenfeld um eine ungerade Anzahl von Stellen nach rechts verschoben wird.

Für die soeben besprochenen sieben verschiedenen Instruktionen wollen wir jetzt einige Beispiele folgen lassen.

Beispiele:

Vorher:

Register 1	0000 4800
Speicherplätze 004 800	12 34 56 78 90 12 3C
bis 004 806	
Speicherplätze 004 810	11 22 33 44 55 66 7D
bis 004 816	
Speicherplätze 004 900	00 00 01 23 4C
bis 004 904	
Speicherplätze 004 905	00 34 12 00 0D
bis 004 909	
Speicherplätze 004 90A	00 00 00 00 00 00 03 10
bis 004 914	00 40 0D
Speicherplätze 004 915	00 00 68 26 14 00 13 01
bis 004 91F	35 00 1D
Speicherplätze 004 920	77 77 77 77 77 77 77 77
bis 004 92A	77 77 77

1. Instruktion: SP FB 4 4 1 100 1 105

Nach dieser Instruktion:

Speicherplätze 004 900	00 34 13 23 4C
bis 004 904	

2. Instruktion: ZAP F8 A 4 1 120 1 100

Nach dieser Instruktion:

Speicherplätze	004 920	00 00 00 00 00 00 00 00
bis	004 92A	01 23 4C

3. Instruktion: MP FC A 4 1 115 1 105

Nach dieser Instruktion:

Speicherplätze	004 90A	00 00 00 10 57 73 36 48
bis	004 914	00 00 0C

4. Instruktion: DP FD A 4 1 115 1 105

Nach dieser Instruktion:

Speicherplätze	004 915	20 00 62 72 01 9C
bis	004 91F	00 13 07 00 1D

5. Instruktion: MVO F1 6 5 1 010 1 000

Nach dieser Instruktion:

Speicherplätze	004 810	01 23 45 67 89 01 2D
bis	004 816	

6. Instruktion: MVO F1 4 5 1 012 1 000

Nach dieser Instruktion:

Speicherplätze	004 810	11 22 45 67 89 01 2D
bis	004 816	

Zum Schluß dieses Abschnittes wollen wir noch eine Instruktion kennenlernen, in die nur ein einziger Operand verwickelt ist, obwohl auch hier das SS-Format zugrunde liegt. Die sechs Bytes, die eine Instruktion im SS-Format benötigt, besitzen bei dieser Instruktion die in der Abb. 4.11. gezeigte Bedeutung.

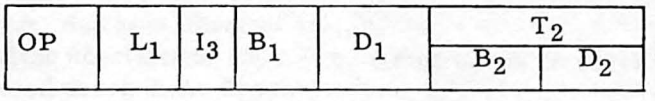


Abb. 4.11. SS-Format für die Instruktion „Schiften und Runden"

Die zu besprechende Instruktion lautet:

Schiften und Runden dezimal — Shift and Round Decimal	F0	SRP	SS

Obwohl diese Instruktion im SS-Format vorliegt, wird von ihr nur der erste Operand betroffen. Der Instruktionsteil I_3 (siehe Abb. 4.11.) muß einen Wert im Wertebereich von 0 bis einschließlich 9 aufweisen. Die effektive Adresse E_2 wird nicht gebildet, so daß durch B_2 und D_2 auch kein Hauptspeicherplatz angesprochen wird. Es wird vielmehr der Inhalt der Bitstellen 26 bis 31 von T_2, d.h. die sechs am weitesten rechts liegenden Bitstellen, als vorzeichenbehaftete ganzzahlige Dualzahl angesehen. Enthält Bitstelle 26 eine 1, so gilt diese Zahl als negativ, bei einer 0 als positiv.

Enthält die Bitstelle 26 eine 0, so werden die Ziffern des ersten Operanden um so viele Ziffernstellen nach links verschoben wie der Wert angibt, der in den Bitstellen

26 bis 31 von T_2 vorliegt. Nullen werden in der erforderlichen Anzahl auf die freiwerdenden rechten Stellen nachgezogen. Der Inhalt von I_3 ist nicht in die Operation verwickelt, doch muß er im Wertebereich [0,9] liegen. Der Bedingungsschlüssel wird auf 0,1 oder 2 in Abhängigkeit vom Resultat der Operation gesetzt. Wenn beim Schiften (Stellenversetzen) nach links eine von Null verschiedene Dezimalziffer verlorengeht, wird der Bedingungsschlüssel in üblicher Weise auf 3 gesetzt und zeigt damit einen Überlauf an.

Enthält die Bitstelle 26 von T_2 eine 1, so werden die Dezimalziffern des ersten Operanden nach rechts verschoben. Der Umfang des Schiftens ist durch den Wert des Zweierkomplements der auf den Bitstellen 26 bis 31 von T_2 stehenden Dualzahl gegeben. In die links freiwerdenden Stellen des ersten Operanden werden Nullen eingeschleust. Zur letzten Ziffer, die beim Schiften nach rechts gerade verschwindet, wird der Inhalt von I_3 addiert. Ein bei dieser Addition evtl. entstehender Übertrag pflanzt sich nach links fort, so daß der dem Rechtsschift unterliegende erste Operand auf diese Weise gleichzeitig gerundet wird. Beim Rechtsschift können nur die Bedingungsschlüssel 0,1 oder 2 auftreten.

Die Instruktion SRP kommt infolge dieser Wirkungsweise dem Versetzen des Dezimalpunktes gleich, d.h. einer Multiplikation der ersten Operanden mit einer Potenz von 10. Beim Verschieben des Dezimalpunktes nach rechts wird gleichzeitig ein Runden vorgenommen. Um den üblichen Rundungsregeln zu genügen, wird man wohl I_3 vorwiegend auf 5 setzen. Für diese Instruktion sollen nunmehr noch zwei Beispiele aufgeführt werden.

Beispiele:

Vorher:

Register 1		0000 4800
Speicherplätze	004 810	11 22 33 44 55 66 7D
bis	004 816	
Speicherplätze	004 900	00 00 01 23 4C
bis	004 904	

1. Instruktion: SRP F0 6 5 1 010 0 03D

Nach dieser Instruktion:

Speicherplätze	004 810	00 01 12 23 34 45 6D
bis	004 816	

In die durch diese Instruktion bewirkte Operation geht ein sieben Bytes langes Feld (L_1=6) ein; der der Rundung zugrundeliegende Wert ist der auf I_3 stehende, nämlich 5. Die Verschiebung der Stellen erfolgt nach rechts, da die Bitstellen 26 bis 31 von T_2 den dezimalen Wert -3 enthalten (dual 11 11 01). Die letzte hinausgeschobene Dezimalziffer ist also die 6 auf der dritten Ziffernstelle von rechts. Diese zu I_3 addiert, ergibt 11; der Übertrag bewirkt, daß die 5 auf der vierten Ziffernstelle von rechts auf 6 aufgerundet wird.

2. Instruktion: SRP F0 4 5 1 100 0 003

Nach dieser Instruktion:

Speicherplätze	004 900	00 12 34 00 0C
bis	004 904	

Durch diese Instruktion erfolgt eine Stellenverschiebung um 3 Stellen nach links im fünf Bytes langen Feld, das die Hauptspeicherplätze 004 900 bis 004 904 belegt.

4.10. Programmunterbrechungen

Mitunter ist es notwendig, die Ausführung eines Programmes aufzuschieben, weil vielleicht ein Programm höherer Priorität zunächst den Computer benutzen muß. Es ist auch denkbar, daß irgendetwas im Programm geschehen ist, so daß es unmöglich wird, es weiter fortzusetzen; beispielsweise könnte das Programm auf einen ungültigen Operationsschlüssel gestoßen sein. Mit Ausnahme sehr kleiner Computer ist aber die Computerzeit viel zu kostspielig, um in solchen Fällen erst den Operateur [1] zu veranlassen, regulierend einzugreifen. Die IBM /370 ist mit einer Unterbrechungseinrichtung ausgestattet, wodurch es möglich ist, bestimmte aufgetretene Bedingungen in geeigneter Weise zu behandeln.

Um zu sehen, wie diese Unterbrechungseinrichtung arbeitet, wollen wir annehmen, daß sich im Hauptspeicher des Computers zwei Programme befinden. Bei dem einen soll es sich dabei um ein umfangreiches ingenieurtechnisches Programm handeln, das eine Karte einliest, mit den Werten aus der Karte langandauernde Berechnungen durchführt und anschließend Resultate auf eine andere Karte stanzt; dieser Vorgang läuft bis zur letzten Datenkarte zyklisch ab. Bei dem anderen Programm soll es sich um ein relativ kleines Programm handeln, das fortwährend Sätze von Magnetbändern einliest, einen Teil der in ihnen enthaltenen Informationen druckfertig aufbereitet und diese auf eine Zeile druckt. Im Gegensatz zum ersten Programm benötigt dieses Programm verschwindend geringe Verarbeitungszeiten, doch muß es ständig auf die Ausführung des vorangegangenen Druckvorganges warten, weil die mechanischen Einrichtungen des Druckers nicht die hohen Geschwindigkeiten zulassen, die intern durch die Elektronik erreicht werden. Das erste Programm benötigt, wie schon festgestellt, nur geringe Eingabe/Ausgabe-Zeiten, dagegen hohe Verarbeitungszeiten. Das zweite Programm möge außerdem weitaus wichtiger als das erste Programm sein (durch dieses Programm könnten vielleicht Lohnlisten gedruckt werden, deren Fertigstellung termingebunden ist); also muß es mit Vorrang ablaufen. Es besteht jedoch kein Grund, warum das erste Programm nicht während der Zeiten arbeiten sollte, in denen das zweite Programm auf den viel langsameren Drucker warten muß. Damit der Wechsel zwischen den beiden Programmen anstandslos erfolgen kann, muß es immer dann zu einer Unterbrechung kommen, wenn der Drucker das Drucken einer Zeile beendet hat, damit in diesem Fall die Programmablaufsteuerung sofort wieder dem zweiten Programm übergeben werden kann. Hat das zweite Programm alle Verarbeitungsinstruktionen ausgeführt, die während der Zeitlücke des ersten Programmes gerade ablaufen können, so muß es selbst die Programmablaufsteuerung wieder zurück an die nächste auszuführende Instruktion des ersten Programmes übergeben können. Damit dieses Unterbrechungssystem einwandfrei arbeiten kann, ist es notwendig, daß der Zustand des Computersteuerwerks exakt festgehalten wird, wenn es zu einer Unterbrechung kommt, damit das erste Programm später korrekt wieder aufgenommen werden kann. Die zu einer Wiederaufnahme des unterbrochenen Programmes erforderlichen Informationen werden ständig in einem Doppelwort fortgeschrieben. Dieses bezeichnet man als Programmstatuswort, abgekürzt PSW. Das Programmstatuswort wollen wir im nächsten Abschnitt 4.11. eingehender besprechen.

Bei der IBM /370 sind sechs verschiedene Arten von Unterbrechungen vorgesehen. Mit jeder Unterbrechungsart sind zwei Doppelwörter verknüpft, und zwar das alte PSW und das neue PSW. Die Speicherplätze für diese Wörter sind fest und unveränderlich. Sie sind in der nachstehenden Tabelle aufgeführt (Abb. 4.12.).

1) Als Berufsbezeichnung für das Maschinenbedienungspersonal gilt Operateur.

In den Klammern sind jeweils die Adressen sedezimal angegeben, während vor den Klammern die Adressen dezimal aufgeführt sind.

Unterbrechungsart	Altes PSW	Neues PSW
Wiederanlauf (Restart)	8 (8)	0 (0)
Externe Unterbrechungen	24 (18)	88 (58)
Supervisor-Aufruf	32 (20)	96 (60)
Programmunterbrechungen	40 (28)	104 (68)
Maschinenfehler-Unterbrechungen	48 (30)	112 (70)
Eingabe/Ausgabe-Unterbrechungen	56 (38)	120 (78)

Abb. 4.12. Alte und neue Programmstatuswörter

Wenn immer es zu einer Unterbrechung kommt, so wird ein Code, der die Unterbrechungsursache (z. B. der Drucker hat seine Arbeit beendet) festhält, in das PSW gesetzt und anschließend wird das laufende PSW auf den Speicherplatz des alten PSW weggespeichert.

Dann werden vom Computer die Informationen vom Platz des neuen PSW hergenommen und als neues laufendes PSW verwendet, mit dessen Hilfe dann das Unterbrechungsprogramm abzulaufen beginnt. Nachdem die Unterbrechung bedient worden ist, wird das Unterbrechungsprogramm auf eine Instruktion geführt, durch die dem Computer aufgetragen wird, vom Speicherplatz 56_{10} oder 40_{10} oder einem anderen geeigneten Speicherplatz das zuvor weggespeicherte PSW zu laden. Damit wird erreicht, daß der Computer wieder in den Arbeitszustand zurückversetzt wird, den er zum Zeitpunkt der Unterbrechung besaß.

Zu Maschinenfehlerunterbrechungen kommt es, wenn durch spezielle in den Computer eingebaute Prüfschaltungen ermittelt wird, daß irgendein Schaltkreis des Computers fehlerhaft arbeitet. Gewöhnlich hat das zur Folge, daß wir das Wartungspersonal rufen müssen, das durch den Ablauf besonderer Diagnostikprogramme versucht, den Fehler baldmöglichst zu finden und zu beheben. Es gibt leichte und schwere Maschinenfehler. Unter leichteren versteht man solche, bei denen die Prüfschaltungen nicht nur den Fehler entdecken, sondern darüberhinaus es auch fertig bringen, die Funktionsfähigkeit vollständig wiederherzustellen und die Instruktion korrekt auszuführen. Eine Maschinenfehlerunterbrechung geschieht erst nach Vollendung der Instruktion, so daß dem Programm das Maschinenproblem mitgeteilt werden kann. Bei einem schweren Maschinenfehler sind die Schaltkreise nicht mehr in der Lage, eine Wiederherstellung so weit vorzunehmen, daß die Instruktion, die gerade in Ausführung befindlich war, weiter ablaufen kann. Das Programm ist damit „gestorben"; das beste, was man hier veranlassen kann, ist, dafür zu sorgen, daß der Computer so bald wie möglich wieder repariert wird.

Programmunterbrechungen treten immer dann auf, wenn ein Programmfehler festgestellt wird, z. B. ungültiger Operationsschlüssel, nichtexistierende Hauptspeicheradresse usw. Sie treten aber auchdnn auf, wenn irgendeine anormale Datenbedingung festgestellt wird, z. B. arithmetischer Überlauf, Versuch einer Division durch Null usw.

Eingabe/Ausgabe-Unterbrechungen (EA-Unterbrechungen) treten immer dann auf, wenn ein EA-Gerät eine ihm aufgetragene Operation beendet hat oder wenn bei einem

EA-Gerät irgendeine besondere Bedingung vorliegt, auf die die Aufmerksamkeit des Computers gelenkt werden muß.

Der durch einen Supervisoraufruf bedingten Unterbrechung liegt eine spezielle Instruktion zugrunde. Wir werden diese ausführlich im Abschnitt 7.9. behandeln.

Zu einer externen Unterbrechung kommt es in folgenden drei Fällen:

a) Eine Echtzeituhr (Zeitgeber) im Computer hat ermittelt, daß eine vorher eingestellte Zeit abgelaufen ist.

b) Der Operateur (Maschinenbediener) drückt die Unterbrechungstaste am Konsol.

c) Durch irgendein externes Gerät (das kann z.B. auch ein anderer Computer sein) wird dem Computer ein Signal gesendet, das vom Computer verlangt, daß er darauf reagiert.

Zu einer Restartunterbrechung kommt es immer dann, wenn der Operateur die Restarttaste auf der Konsole des Computers drückt. Bei einigen Modellen der Computerfamilie IBM /370 können zwei Computer so miteinander verbunden werden, daß ein spezielles Signal des einen dazu benutzt werden kann, um eine Restartunterbrechung beim anderen zu verursachen. Man sagt, daß derart verbundene Computer eine Multiprozessor - Konfiguration bilden.

4.11. Das Programmstatuswort (PSW)

Das Programmstatuswort (PSW) besitzt den in Abb. 4.13. dargestellten Aufbau.

Bitstellen	0	1	2	3	4	5	6	7	8 bis 11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Inhalt bei der BS	Kanalmaske						E A	E	Speicherschutz-Schlüssel	A	M	W	P	Unterbrechungsschlüssel															
Inhalt bei der ES	0	R	0	0	0	T	E A	E	Speicherschutz-Schlüssel	A	M	W	P	0	0	BS	Programm-Maske					0	0	0	0	0	0	0	0

Bitstellen	32	33	34	35	36 bis 39	40 bis 63
Inhalt bei der BS	ILS		BS		Programm-Maske	Instruktionsadresse
Inhalt bei der ES	0	0	0	0	0 0 0 0	Instruktionsadresse

Abb. 4.13. Der Aufbau des Programmstatuswortes (PSW)

Ist auf der Bitstelle 12 des PSW eine Null vorhanden, so sagen wir, daß das PSW ein PSW der Basissteuerung (BS) [1] ist. Bei einer 1 auf der Bitstelle 12 liegt ein PSW der erweiterten Steuerung (ES) [2] vor.

Die folgenden Ausführungen beschäftigen sich mit einer Übersicht über die Bedeutung der einzelnen Felder im PSW. Es sei dabei darauf hingewiesen, daß die Bitstellen 0 bis 7 oft gemeinsam als Systemmaske bezeichnet werden.

1) Im angloamerikanischen Sprachraum spricht man von „basic control", abgekürzt BC.
2) Im angloamerikanischen Sprachraum spricht man von „extended control", abgekürzt EC

1. Kanalmasken (Bitstellen 0 bis 5), nur im BS-Modus
   Jede dieser Bitstellen erlaubt gewisse EA-Unterbrechungen, wenn sie eine 1 enthält. Bei einer 0 ist die betreffende Unterbrechungsart gesperrt.

2. EA-Bitstelle (Bitstelle 6)
   a) ES-Modus
      Enthält diese Bitstelle eine 0, so werden alle EA-Unterbrechungen verhindert. Bei einer 1 sind individuelle Unterbrechungen erlaubt oder verboten, je nachdem, welchen Inhalt die Bitstellen des Steuerregisters 2 aufweisen.
   b) BS-Modus
      In diesem Modus steuert der Inhalt dieser Bitstelle alle EA-Unterbrechungen, soweit diese nicht durch die Kanalmaskenbitstellen überwacht werden.

3. Bitstelle E für externe Unterbrechungen (Bitstelle 7)
   Enthält diese Bitstelle eine 0, so sind alle externen Unterbrechungen blockiert. Bei einer 1 auf dieser Bitstelle kommen diejenigen externen Unterbrechungen zum Zuge, bei denen die entsprechende Bitstelle im Steuerregister 0 auf 1 steht.

4. Speicherschutzschlüssel (Bitstellen 8 bis 11)
   Befindet sich zur gleichen Zeit mehr als ein Programm im Hauptspeicher, so können diesen Bitstellen für jedes Programm unterschiedliche Werte zugeordnet werden. Dadurch kann verhütet werden, daß ein Programm Zugriff zu den Daten eines anderen Programmes gewinnt.

5. Bitschalter AMWP (Bitstellen 12 bis 15)
   a) Bitstelle 12 (A)
      Eine 1 zeigt im BS-Modus an, daß der Computer im USASCII-Code arbeitet, eine 0, daß er im EBCDIC arbeitet. Im ES-Modus enthält diese Bitstelle stets eine 1.
   b) Bitstelle 13 (M)
      Ist diese Bitstelle auf 0 gesetzt, so werden Maschinenfehler ignoriert; dieser Status wird dann und nur dann gefahren, wenn das Wartungspersonal mit der Wartung oder Reparatur der Maschine beschäftigt ist. Enthält diese Bitstelle eine 1, so werden Maschinenfehler so behandelt, wie es die Inhalte der Steuerregister 14 und 15 besagen.
   c) Bitstelle 14 (W)
      Bitstelle 14 steht auf 1, wenn sich die Maschine im Wartestatus befindet; weist diese Bitstelle eine 0 auf, so befindet sich die Maschine im Laufstatus. Im Laufstatus werden Instruktionen ausgeführt, im Wartestatus dagegen nicht. In beiden Statusarten können Unterbrechungen aufgenommen werden. Tatsächlich kann der Wartestatus nur durch eine Unterbrechung verlassen werden. Im allgemeinen wird der Computer dann in den Wartestatus versetzt, wenn das ablaufende Programm einen Punkt erreicht hat, von dem aus es erst dann wiederaufgenommen werden kann, wenn eine Unterbrechung eingetreten ist.
   d) Bitstelle 15 (P)
      Bitstelle 15 enthält eine 1, wenn sich der Computer im Problemstatus befindet; enthält sie eine 0, so befindet sich der Computer im Supervisorstatus. Im Problemstatus können nur die Verarbeitungsinstruktionen ausgeführt werden, doch gewisse privilegierte Instruktionen, wozu alle EA-Instruktionen gehören, sind ungültig. Diese privilegierten Instruktionen können nur im Supervisorstatus ausgeführt werden. Die Verwendung dieser beiden Statusarten erleichtert die Fehlersuche bzw. die Fehlerkorrektur in neuen Programmen. Keine der bisher behandelten Instruktionen ist eine privilegierte.

6. Unterbrechungsschlüssel (Bitstellen 16 bis 31)
   a) BS-Modus
   Die Bitstellen 16 bis 31 enthalten den Unterbrechungsschlüssel. Ihr Inhalt gibt uns Aufschluß über den Grund der Unterbrechung.
   b) ES-Modus
   Im ES-Modus wird der Unterbrechungsschlüssel nicht im PSW gespeichert. Vielmehr werden hierzu bestimmte Hauptspeicherstellen herangezogen, und zwar die folgenden:

Sedezimaladresse(n)	86 und 87	8A und 8B	8E und 8F	AE und AF
Schlüssel	Externe Unterbrechungen	Supervisoraufruf-Unterbrechungen	Programmunterbrechungen	EA-Unterbrechungen

   Das Feld für den Unterbrechungsschlüssel wird nicht benutzt für Restart- und Maschinenfehlerunterbrechungen.

7. Instruktionslängenschlüssel (Bitstellen 32 und 33)
   a) BS-Modus
   Die Bitstellen 32 und 33 enthalten den Instruktionslängenschlüssel (ILS). Durch den ILS wird die Anzahl der Halbworte festgehalten, die die zuletzt ausgeführte Instruktion im Hauptspeicher belegt. Er steht also auf 01 bei Instruktionen im RR-Format, auf 10 bei Instruktionen im RX-Format und auf 11 bei Instruktionen im SS-Format. Ein Schlüssel von 00 zeigt an, daß vom Computer keine Instruktionslänge festgestellt wurde.
   b) ES-Modus
   Der Instruktionslängenschlüssel wird nicht im PSW gespeichert. Er erscheint vielmehr auf dem Speicherplatz mit der sedezimalen Adresse 88 bei einer Supervisoraufruf-Unterbrechung bzw. auf dem Speicherplatz mit der sedezimalen Adresse 8D bei einer Programmunterbrechung. Bei anderen Unterbrechungen wird der Instruktionslängenschlüssel nicht gespeichert.

8. Bedingungsschlüssel (Bitstellen abhängig vom Steuermodus)
   Die Bitstellen 34 und 35 enthalten den Bedingungsschlüssel (BS) im BS; im ES ist dieser dagegen auf den Bitstellen 18 und 19 zu finden.

9. Programm-Maske (Bitstellen abhängig vom Steuermodus)
   Die Programm-Maske belegt in den beiden Steuermodi unterschiedliche Bitstellen im PSW. Im BS-Modus sind es die Bitstellen 36 bis 39, im ES-Modus die Bitstellen 20 bis 23. Durch den Inhalt dieser Bitstellen ist festgelegt, ob gewisse ungewöhnliche Datenbedingungen ignoriert werden sollen oder ob sie zu einer Unterbrechung Veranlassung geben sollen. Enthält die betreffende Maskenbitstelle eine 1, so erfolgt beim Auftreten der entsprechenden Unterbrechungsbedingung eine Unterbrechung; enthält die Maskenbitstelle dagegen eine 0, so werden die Unterbrechungsbedingungen ignoriert, so gehen verloren.
   a) Bitstelle 20 (ES) bzw. Bitstelle 36 (BS)
   Diese Bitstelle regelt die Behandlung eines Festpunktüberlaufs, der als Resultat einer arithmetischen Operation mit dualen Zahlen in Festpunktdarstellung passieren kann.
   b) Bitstelle 21 (ES) bzw. Bitstelle 37 (BS)
   Diese Bitstelle regelt die Reaktion auf einen Dezimalüberlauf, zudem es bei arithmetischen Operationen mit gepackten Dezimalzahlen kommen kann.

c) Bitstelle 22 (ES) bzw. Bitstelle 38 (BS)
Diese Bitstelle regelt die Behandlung eines Exponentenunterlaufs. Zu diesem kann es kommen, wenn bei einer arithmetischen Operation mit Zahlen in Gleitpunktdarstellung ein Ergebnis entsteht, dessen Charakteristik kleiner als Null ist, d.h. der Exponent ist kleiner als -64.

d) Bitstelle 23 (ES) bzw. Bitstelle 39 (BS)
Diese Bitstelle regelt die Behandlung eines Signifikanzfehlers. Dieser kann als Ergebnis einer arithmetischen Operation mit Zahlen in Gleitpunktdarstellung auftreten. Man versteht darunter, daß eine Zahl in Gleitpunktdarstellung vorliegt, deren sämtliche Mantissenstellen Nullen aufweisen, deren Charakteristik jedoch ungleich Null ist.

10. Instruktionsadresse (Bitstellen 40 bis 63)
Die Bitstellen 40 bis 63 enthalten schließlich die Adresse der nächsten auszuführenden Instruktion.

Im Basissteuermodus (BS) befinden sich alle Informationen, die auf ein Programm bezug nehmen, das sich in Ausführung befindet, im PSW, abgesehen vom Inhalt der Register. Jedes von einer IBM /370 ausgeführte Programm benötigt ein PSW.

Im erweiterten Steuermodus (ES) befinden sich die zu einem Programm gehörenden Informationen nicht nur im PSW, sondern auch in Steuerregistern und auf bestimmten Hauptspeicherstellen, die alle im unteren Teil des Hauptspeichers, d.h. im Teil mit den niedrigen Adressen, liegen. Die Informationen in den spezifizierten Hauptspeicherplätzen werden jedoch dann und nur dann benötigt, wenn es wirklich zu einer Unterbrechung gekommen ist.

Aufgabengruppe 10

1) Auf den Hauptspeicherplätzen, die bei der sedezimalen Adresse 0230FE beginnen, ist ein Programm gespeichert, das aus der nachstehenden Instruktionsfolge besteht.

Adressen		Instruktionen
Ø23 ØFE	BALR	Ø5 B Ø
Ø23 1ØØ	-	READ Ø234ØØ,Ø232ØØ
Ø23 11Ø	MVC	D2 Ø 2 B 4Ø1 B 3ØØ
Ø23 116	MVC	D2 Ø 2 B 4Ø5 B 3Ø3
Ø23 11C	MVC	D2 Ø 1 B 4Ø9 B 3Ø6
Ø23 122	MVC	D2 Ø Ø B 4ØØ B 2ØØ
Ø23 128	MVC	D2 Ø Ø B 4Ø4 B 2Ø2
Ø23 12E	MVC	D2 Ø Ø B 4Ø8 B 2Ø3
Ø23 134	-	PRINT Ø235ØØ
Ø23 144	BC	47 F Ø B ØØØ
Ø23 2ØØ	-	EOJ

Außerdem sind folgende Daten im Hauptspeicher gespeichert:

Adressen	Inhalt
Ø23 4ØØ bis Ø23 44F	Leerzeichen, d.h. 4Ø(sed) auf jeder Stelle
Ø23 5ØØ bis Ø23 577	Leerzeichen, d.h. 4Ø(sed) auf jeder Stelle
Ø23 3ØØ bis Ø23 3Ø3	5B 4Ø 6B 4B (sedezimal)

Was wird auf den einzelnen Zeilen gedruckt, wenn die Spalten 1 bis 8 der Eingabekarten folgende Informationen enthalten?

a) 12345678 b) 00134598 c) 00000678 d) 00000000

2) Ändere das Programm von Aufgabe 1) so ab, daß an Stelle des Operationsschlüssels 47 der Operationsschlüssel 07 verwendet werden kann!

3) Ändere das Programm von Aufgabe 1) so ab, daß eine Leerzeile gedruckt wird, wenn die Spalten 1 bis 8 einer Eingabekarte lauter Nullen enthalten; siehe hierzu Fall d)

4) Ändere das Programm von Aufgabe 1) so ab, daß ab dem Hauptspeicherplatz mit der Adresse 023 110 die folgenden Instruktionen stehen:

Adressen		Instruktionen
Ø23 11Ø	PACK	F2 7 3 B 21Ø B 3ØØ
Ø23 116	PACK	F2 7 3 B 218 B 3Ø4
Ø23 11C	AP	FA 7 7 B 21Ø B 218
Ø23 122	UNPK	F3 A 7 B 4ØØ B 21Ø
Ø23 128	BC	47 F Ø B Ø34

Es sollen vom so entstandenen neuen Programm die gleichen Datenkarten wie bei der Aufgabe 1) gelesen werden. Wie sehen die Zeilen aus, die jetzt gedruckt werden?

5) Vom Programm der Aufgabe 4) sollen die bei 023 110 beginnenden Instruktionen wie folgt abgeändert werden:

Adressen	Instruktionen	
Ø23 11Ø	PACK	F2 7 3 B 21Ø B 3ØØ
Ø23 116	PACK	F2 7 3 B 218 B 3Ø4
Ø23 11C	CVB	4F 3 Ø B 21Ø
Ø23 12Ø	CVB	4F 4 Ø B 218
Ø23 124	AR	1A 3 4
Ø23 126	CVD	4E 3 Ø B 21Ø
Ø23 12A	UNPK	F3 A 7 B 4ØØ B 21Ø
Ø23 13Ø	BC	47 Ø Ø Ø ØØØ

Wie beeinflußt diese Änderung die Ausgabe, verglichen mit der Aufgabe 3)?

6) Schreibe ein Programm, das die Datenkarten von Aufgabe 1) liest. Nach dem Lesen der letzten Karte ist eine Zeile zu drucken, auf der die Summe aller Zahlen erscheint, die auf den Spalten 5 bis 8 der einzelnen Karten abgelocht waren. Zur Bildung der Summe ist die Instruktion AP einzusetzen!

7) Das Programm von Aufg. 6) ist neu abzufassen. Dabei sollen Instruktionen der Dezimalarithmetik nicht benutzt werden.

8) Auf den Hauptspeicherplätzen 022 00 bis 022 003 steht eine ganzzahlige Dualzahl. Schreibe eine Folge von Instruktionen, beginnend beim Hauptspeicherplatz mit der Adresse 021 E00, die diese ganze Zahl durch 4 ohne Rest teilbar macht, indem je nach Bedarf die Zahlen 0, 1, 2 oder 3 zu dieser gespeicherten Zahl addiert werden.

9) Schreibe ein Programm, das Karten einliest, die wie folgt abgelocht sind:

Spalten 1 bis 7:	xxxxxxx
Spalten 8 bis 14:	yyyyyyy
Spalten 15 bis 80:	leer

Es sei vorausgesetzt, daß xxxxxxx und yyyyyyy numerische Werte darstellen. Die eingelesenen Werte sollen wie folgt auf neue Karten gestanzt werden:

Spalten 1 bis 7:	xxxxxxx
Spalten 8 bis 20:	leer
Spalten 21 bis 27:	yyyyyyy
Spalten 28 bis 80:	leer

10) Aufgabe 9) ist so abzuändern, daß auf die Ausgabekarten außerdem auf die Spalten 30 bis 38 die Differenz aus den x- und y-Werten gestanzt wird. Außerdem soll es möglich sein, negative Zahlen einzugeben. (Eine Lochung in Zeile 11 auf den Einerstellen der beiden Felder soll die betreffende Zahl als negativ ausweisen.)

11) Schreibe ein Programm, das Karten einliest, auf denen in den Spalten 1 bis 3 beliebige ein-, zwei- oder dreistellige Zahlen, jeweils auf Spalte 1 beginnend, abgelocht sind. Enthalten die abgelochten Zahlen weniger als drei Ziffern, so sollen sich nach rechts Leerzeichen anschließen. Ausgedruckt werden sollen die gleichen Zahlen, doch sollen diese in einem dreistelligen Feld rechtsbündig stehen, d.h. führende Nullen sollen bei Bedarf eingefügt werden. Beispiele:

Eingabe	Ausgabe
378	378
37	037
3	003

5. Die Programmierung der Eingabe/Ausgabe

5.1. Grundlagen der Eingabe/Ausgabe

Unter der Eingabe versteht man die Übertragung von Informationen von einer externen Quelle in den Hauptspeicher des Computers. Unter der Ausgabe versteht man die Übertragung von Informationen aus dem Hauptspeicher zu einem externen Ziel.

Alle Maschinen, die zur Übertragung von Informationen zwischen dem Hauptspeicher der IBM /370 und einem externen Medium dienen, werden als Eingabe/Ausgabe-Geräte (kurz: EA-Geräte) bezeichnet. Die Geräte, die an die IBM /370 am häufigsten angeschlossen werden, sind Lochkartenleser, Lochkartenstanzer, Zeilendrucker, Schreibmaschinen, Datensichtgeräte, Magnetbandgeräte und Magnetplattenspeichergeräte. Eine sogenannte Steuereinheit steuert die Arbeitsweise jedes EA-Gerätes. In einigen Fällen ist die Steuereinheit unmittelbar in das EA-Gerät eingebaut und kann deshalb physisch nicht von diesem unterschieden werden. Bei anderen Geräten dagegen ist die Steuereinheit vollständig vom EA-Gerät getrennt, stellt also physisch ein eigenes Gerät, das Steuergerät, dar. Ein Steuergerät kann dabei durchaus mehrere EA-Geräte steuern; namentlich bei Magnetband- und Magnetplattenspeichergeräten ist das der Fall. Alle physikalischen Vorgänge, wie das Stanzen von Löchern in Lochkarten oder das Zurückspulen von Magnetbändern, werden von den EA-Geräten selbständig durchgeführt. Die Entscheidung darüber, welche Löcher zu stanzen sind bzw. welcher Satz auf ein sich bewegendes Magnetband aufgezeichnet werden soll, obliegt den Steuereinheiten. Im allgemeinen kann man sagen, daß alle rein physikalischen Bewegungen von den EA-Geräten selbst ohne eingehende Überwachung durch die Steuereinheiten ausgeführt werden. Jeder Vorgang jedoch, der mit einer Datenübertragung verknüpft ist, erfordert die Mitwirkung und die beständige Kontrolle durch die Steuereinheit.

Es gibt einige Probleme, mit denen man sich beim Anschluß von EA-Geräten an einen Computer auseinandersetzen muß. Da ist vor allem das Zeitproblem zu erwähnen. Der schnellste verfügbare Kartenleser kann z.B. ein Byte in 625 μs in den Hauptspeicher übertragen. Vergleicht man diese Zeit mit der Zugriffszeit zu einem Byte im Hauptspeicher, die beim langsamsten Modell bei 0,47 μs liegt, so kommt man zu folgender interessanten Feststellung: Wenn der Computer die Verarbeitung unterdrücken bzw. aufschieben muß, um auf den Kartenleser zu warten, dann fallen für den Hauptspeicher immerhin rd. 99,92 % Totzeiten stets dann an, wenn eine Karte gelesen ist. Früher konstruierte Computertypen wiesen fast immer ein solches Mißverhältnis zwischen interner Verarbeitung und EA auf. Deshalb wurden beim Entwurf der IBM /370 zwei Kriterien herausgestellt, nach denen dann die Anlage konzipiert wurde.

1) Es muß möglich sein, daß die EA gleichzeitig (simultan) mit der internen Verarbeitung abläuft.
2) Es muß möglich sein, daß mehr als eine EA-Operation zur gleichen Zeit stattfinden kann.

Soll die EA gleichzeitig mit internen Operationen ablaufen, dann muß es eine Einrichtung geben, die die EA-Operationen steuert und die bei dieser Steuerung unabhängig vom zentralen Steuerwerk ist. Das wird bei der IBM /370 durch den Kanal erreicht. Ein Kanal stellt in gewisser Hinsicht selbst einen kleinen Computer dar. Er führt ein gespeichertes Programm von sogenannten Befehlen aus, genau so wie

die ZE [1] ein gespeichertes Programm von Instruktionen ausführt. Der einzige Unterschied besteht eigentlich nur darin, daß ein Kanalprogramm sich ausschließlich mit EA-Vorgängen befaßt. (Wir sprechen zwar beim Kanal von Befehlen und bei der ZE von Instruktionen, aber von der Logik her handelt es sich um das gleiche, nämlich um Operationsanleitungen.) Stellt die ZE fest, daß eine EA-Operation ablaufen muß, so teilt sie dem Kanal mit, auf welchem Speicherplatz der erste Befehl des gewünschten Kanalprogramms zu finden ist. Der Kanal führt dann dieses Programm selbständig aus. Wenn der Kanal sein Programm beendet hat (manchmal auch schon, wenn er einen speziellen Programmteil beendet hat), so signalisiert er der ZE, daß er das Ende des Programmes erreicht hat. Dieses Signal hat entweder eine Programmunterbrechung zur Folge oder es kann durch geeignete ZE-Instruktionen überprüft werden.

Der zweite, vorhin erwähnte Punkt, nämlich das gleichzeitige Ablaufen von mehreren EA-Operationen, ist ein wenig komplizierter zu bewerkstelligen. Zwei Lösungen bieten sich an. Für die schnelleren EA-Geräte, wie Magnetband- bzw. Magnetplattenspeichergeräte, besteht die einfachste Lösung darin, mehrere Kanäle vorzusehen. Maximal können an die IBM /370 bis zu vierzehn Kanäle angeschlossen werden, obgleich nicht allzu viele Systeme mit mehr als drei Kanälen ausgerüstet werden. Wird das gleichzeitige (simultane) Arbeiten vieler langsamer EA-Geräte (z.B. Kartenleser) verlangt, so ist die zweckmäßigste Lösung wohl darin zu suchen, daß man einen Kanal in viele *Unterkanäle* unterteilt und jeden Unterkanal an den Einrichtungen und Möglichkeiten des Kanals teilhaben läßt. Wenn alle EA-Geräte langsam genug sind, so kann der Kanal im Wechsel alle Unterkanäle der Reihe nach bedienen. Stehen einige Bytes zur Übertragung bereit, so werden diese übertragen, bevor er zum nächsten Unterkanal übergeht. Kann er auf diese Weise schnell genug alle Unterkanäle auswählen und bedienen, so hat es den Anschein, als ob jedes EA-Gerät über die volle Zeit hinweg alle Möglichkeiten und Einrichtungen des Kanals beansprucht. Ein sogenannter *Bytemultiplexkanal* arbeitet in dieser Weise. Ein anderer Kanaltyp, der sogenannte *Selektorkanal*, besitzt nur einen einzigen Unterkanal und kann somit zu einem Zeitpunkt nur Informationen zu oder von einem einzigen Geräte übertragen. Man sagt, daß Selektorkanäle im *Einpunktbetrieb* (engl.: Burst Mode) arbeiten, während Bytemultiplexkanäle normalerweise verzahnt arbeiten, d.h. im *Mehrpunktbetrieb* (engl.: Interleaved Mode oder Multiplex Mode) [2].

Es ist möglich, daß EA-Geräte mit hoher Übertragungsgeschwindigkeit an einen Bytemultiplexkanal angeschlossen werden. Arbeitet dann ein solches Gerät, so kann seine Geschwindigkeit zu groß für diesen relativ langsamen Kanaltyp werden. Wenn also ein EA-Gerät hoher Geschwindigkeit die Steuereinrichtungen des Kanals beanspruchen will, so muß der Kanal im Bedarfsfall alle anderen Geräte zeitweise aussperren, um sich voll und ganz dem schnellen Gerät widmen zu können; d.h. aber nichts anderes, als daß er in diesem Augenblick in den Einpunktbetrieb übergeht und somit wie ein Selektorkanal arbeitet. Sobald aber die Hochgeschwindigkeitsübertragung erledigt ist, kehrt der Kanal zu seiner regelmäßigen Arbeitsweise, dem Mehrpunktbetrieb, zurück. Diese Operationsart ist, obgleich möglich, im allgemeinen nicht wirtschaftlich. Es gibt deshalb einen dritten Kanaltyp, den sogenannten *Blockmultiplexkanal* [3], der für Operationen mit mehreren EA-Geräten hoher Übertragungsrate entworfen wurde. Gewisse Typen von Magnetplattenspeichergeräten gehören z.B. zu dieser Kategorie. Ein Blockmultiplexkanal fragt seine Unterkanäle wechselweise ab, bis er einen findet, der bereit ist, Daten zu übertragen. Von diesem transportiert er dann

1) ZE ist die Abkürzung von Zentraleinheit (siehe hierzu Abschnitt 3.10.)

2) Man bezeichnet den Mehrpunktbetrieb auch als simultane Arbeitsweise, den Einpunktbetrieb als normale Arbeitsweise.

3) Im angloamerikanischen Sprachraum als „Block Multiplex Channel" bezeichnet.

die Daten bis zum Ende des Datenblockes. Nach erfolgter Datenübertragung geht er sofort zum nächsten Unterkanal über und fährt mit dessen Abfrage fort. Das heißt aber nichts anderes, als daß ein Blockmultiplexkanal während nichtproduktiver Tätigkeiten eines EA-Gerätes, z.B. Suchbewegungen von Zugriffsmechanismen bei Magnetplattenspeichergeräten, freigemacht und für irgendein anderes Kanalprogramm eingesetzt werden kann. An Blockmultiplexkanäle sollten deshalb nur solche EA-Geräte angeschlossen werden, die für gewisse Zeitperioden vom Datentransport ausgeschlossen werden bzw. werden müssen, wozu ohne Zweifel grundsätzlich die Magnetplattenspeichergeräte zählen.

Kanäle werden mit 0, 1, 2, ···, D, E (sedezimal) durchnumeriert. Kanal 0 ist stets ein Bytemultiplexkanal. Die anderen Kanäle sind gewöhnlich eine Mixtur aus Selektor- und Blockmultiplexkanälen. An manche Modelle der Computerfamilie IBM /370 kann ein zweiter Bytemultiplexkanal angeschlossen werden. Jedes an einen Kanal angehängte EA-Gerät besitzt eine Adresse, die ein Byte lang ist.[2] Somit können bis zu 256 EA-Geräte an einen Kanal angeschlossen werden, da acht Bitstellen insgesamt 256 verschiedene Verschlüsselungen zulassen. EA-Geräte, die unter Kontrolle einer gemeinsamen Steuereinheit stehen, müssen lückenlos aufeinanderfolgende Adressen besitzen. Bei einem Bytemultiplexkanal können Steuereinheiten für EA-Geräte hoher Übertragungsgeschwindigkeiten nur an die ersten acht Unterkanäle angeschlossen werden. Diese sind unter der Bezeichnung „gemeinsam benutzte Unterkanäle" bekannt. Einige Arten von Bytemultiplexkanälen weisen solche Unterkanäle nicht auf. Bei anderen Bytemultiplexkanälen wiederum darf nicht mehr als ein EA-Gerät an einen Unterkanal angehängt werden. Lochkartengeräte (Leser und Stanzer) werden gewöhnlich an einen Bytemultiplexkanal angeschlossen. Die Abb. 5.1. zeigt schematisch eine mögliche Konfiguration (Zusammenstellung) eines Datenverarbeitungssystems IBM /370. Es handelt sich hierbei selbstverständlich nur um eine Möglichkeit unter vielen.

Ein EA-Gerät kann an mehr als eine Steuereinheit angeschlossen werden. Ein solches Vorgehen erfordert jedoch, daß das betreffende Gerät eine gesonderte Adresse für jede Steuereinheit besitzt, d.h. jede Steuereinheit unter einer anderen Adresse bekannt ist. Somit beziehen sich z.B. in Abb. 5.1. die Adressen 083 und 0A1 auf das gleiche EA-Gerät; die erste Ziffer spezifiziert hierbei den Kanal. Es ist ebenfalls möglich, daß eine Steuereinheit (und damit die an ihr hängenden EA-Geräte) an mehr als einen Kanal angeschlossen wird. Das wiederum erfordert, daß die EA-Geräte für jeden Kanal unter der gleichen Adresse bekannt sind. Somit beziehen sich in Abb. 5.1. die Adressen 1C0 und 6C0 auf das gleiche Gerät, denn die eigentlichen Geräteadressen C0 sind gleich, die erste Ziffer spezifiziert, wie bereits bekannt, den Kanal selbst.

Bei einigen Modellen der Computerfamilie IBM /370 können gewisse Steuereinheiten direkt in die Zentraleinheit eingebaut werden. In einigen Fällen können ein Kanal und eine Steuereinheit miteinander kombiniert werden; sie bilden dann einen Bauteil innerhalb der Zentraleinheit. Diese Möglichkeiten sind der Kostenreduzierung wegen geschaffen worden, sie berühren in keiner Weise die logischen Wirkungsweisen der Steuereinheiten und der Kanäle.

2) Um ein Gerät vollständig zu adressieren, muß zur Geräteadresse noch die Kanaladresse hinzukommen, die bekanntlich aus vier Dualziffern besteht, so daß also eine EA-Adresse insgesamt 12 Bitstellen belegt.

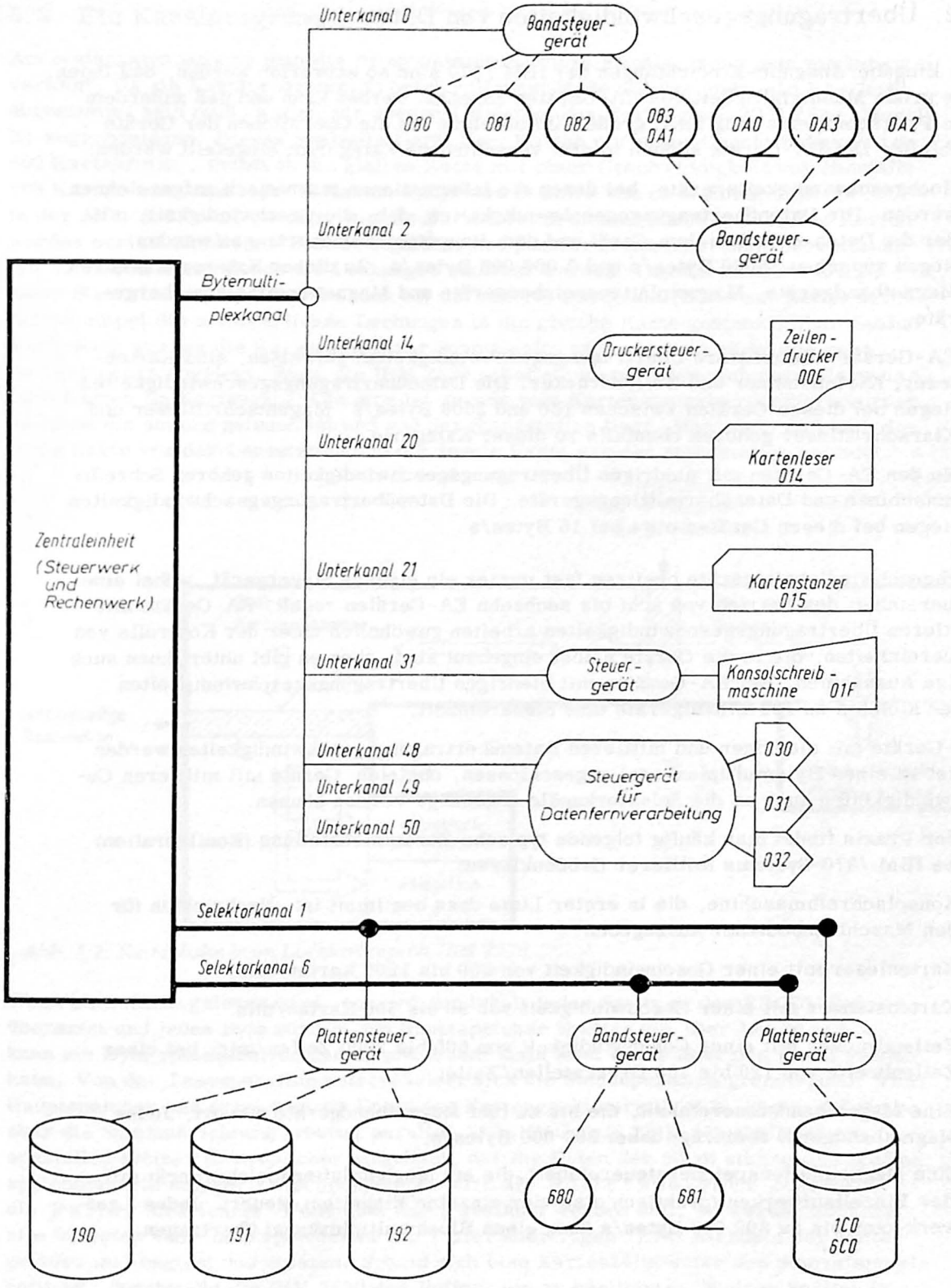


Abb. 5.1. Schema einer möglichen Zusammenstellung einer Datenverarbeitungsanlage IBM /370

5.2. Übertragungsgeschwindigkeiten von Daten

Die Eingabe/Ausgabe-Einrichtungen der IBM /370 sind so entworfen worden, daß ihnen eine große Mannigfaltigkeit von EA-Geräten angepaßt werden kann und daß außerdem dem Programm eine möglichst große Einflußnahme auf die Operationen der Geräte verbleibt. Die EA-Geräte können in drei verschiedene Kategorien eingeteilt werden:

1) Hochgeschwindigkeitsgeräte, bei denen die Informationen magnetisch aufgezeichnet werden. Die Datenübertragungsgeschwindigkeiten, d.h. die Geschwindigkeit, mit der die Daten zwischen dem Gerät und dem Hauptspeicher übertragen werden, liegen zwischen 15000 Bytes/s und 3 000 000 Bytes/s. Zu dieser Kategorie gehören Magnetbandgeräte, Magnetplattenspeichergeräte und Magnettrommelspeichergeräte.

2) EA-Geräte die mittlere Übertragungsgeschwindigkeiten aufweisen, sind Kartenleser, Kartenstanzer und Zeilendrucker. Die Datenübertragungsgeschwindigkeiten liegen bei diesen Geräten zwischen 160 und 2000 Bytes/s. Magnetschriftleser und Klarschriftleser gehören ebenfalls zu dieser Kategorie.

3) Zu den EA-Geräten mit niedrigen Übertragungsgeschwindigkeiten gehören Schreibmaschinen und Datenübermittlungsgeräte. Die Datenübertragungsgeschwindigkeiten liegen bei diesen Geräten etwa bei 15 Bytes/s.

Hochgeschwindigkeitsgeräte besitzen fast immer ein eigenes Steuergerät, wobei eine Steuereinheit den Betrieb von acht bis sechzehn EA-Geräten regelt. EA-Geräte mit mittleren Übertragungsgeschwindigkeiten arbeiten gewöhnlich unter der Kontrolle von Steuereinheiten, die in die Geräte selbst eingebaut sind, aber es gibt unter ihnen auch einige Ausnahmen. Bei EA-Geräten mit niedrigen Übertragungsgeschwindigkeiten teilen sich bis zu 192 Einzelgeräte eine Steuereinheit.

EA-Geräte mit niedrigen und mittleren Datenübertragungsgeschwindigkeiten werden meist an einen Bytemultiplexkanal angeschlossen, obgleich Geräte mit mittleren Geschwindigkeiten auch an die Selektorkanäle angehängt werden können.

In der Praxis findet man häufig folgende typische Zusammenstellung (Konfiguration) eines IBM /370-Systems mittlerer Größenklasse:

a) Konsolschreibmaschine, die in erster Linie dazu bestimmt ist, Nachrichten für den Maschinenbediener auszugeben.

b) Kartenleser mit einer Geschwindigkeit von 600 bis 1200 Karten/min.

c) Kartenstanzer mit einer Geschwindigkeit von 90 bis 500 Karten/min.

d) Zeilendrucker mit einer Geschwindigkeit von 600 bis 2000 Zeilen/min. bei einer Zeilenbreite von 120 bis 160 Druckstellen/Zeile.

e) Eine Magnetbandsteuereinheit, die bis zu fünf Magnetbandgeräte steuert. Jedes Magnetbandgerät überträgt dabei 200 000 Bytes/s.

f) Eine Magnetplattenspeichersteuereinheit, die ein Magnetplattenspeichergerät mit vier Einzellaufwerken (praktisch also vier einzelne Einheiten) steuert. Jedes Laufwerk kann bis zu 800 000 Bytes/s über einen Blockmultiplexkanal übertragen.

5.3. Ein Kanalprogramm

Am einfachsten erlernt man die Programmierung eines Kanals, wenn man ein Beispiel verfolgt. Da ein Kanalprogramm in beträchtlichem Maße auf das spezielle EA-Gerät abgestimmt sein muß, wollen wir unserem Beispiel das Kartengerät IBM 2520, Modell B1 zugrundelegen. Dieses spezielle Gerät liest Karten mit einer Geschwindigkeit von 500 Karten/min., stanzt in die gleiche Karte mit einer Geschwindigkeit von ebenfalls 500 Karten/min. und legt die Karten wahlweise in eines von zwei Ablagefächern ab. In der Abb. 5.2. wird gezeigt, wie die Karten durch dieses Gerät laufen. Die Karten werden seriell (Spalte auf Spalte) zugeführt und passieren zunächst eine Lesestation mit 12 Photozellen, durch die Lochungen entdeckt werden. Danach wandern sie zu einer Stanzstation. In dieser werden bei parallelem Durchlauf (Zeile auf Zeile) durch Stanzstempel die erforderlichen Lochungen in die gleiche Karte gestanzt. Zum Schluß schließlich werden die Karten mit ihrer Schmalseite nach unten in eines von zwei Ablagefächern abgelegt. Wenn die IBM 2520 arbeitet, so befinden sich normalerweise zwei Karten im Kartengang: Die eine ist gerade vom Kartenmagazin zugeführt worden, während die andere gelesen ist und vor der Stanzstation liegt. Man sagt, daß sich die erste Karte vor der Lesestation und die zweite Karte vor der Stanzstation befindet.

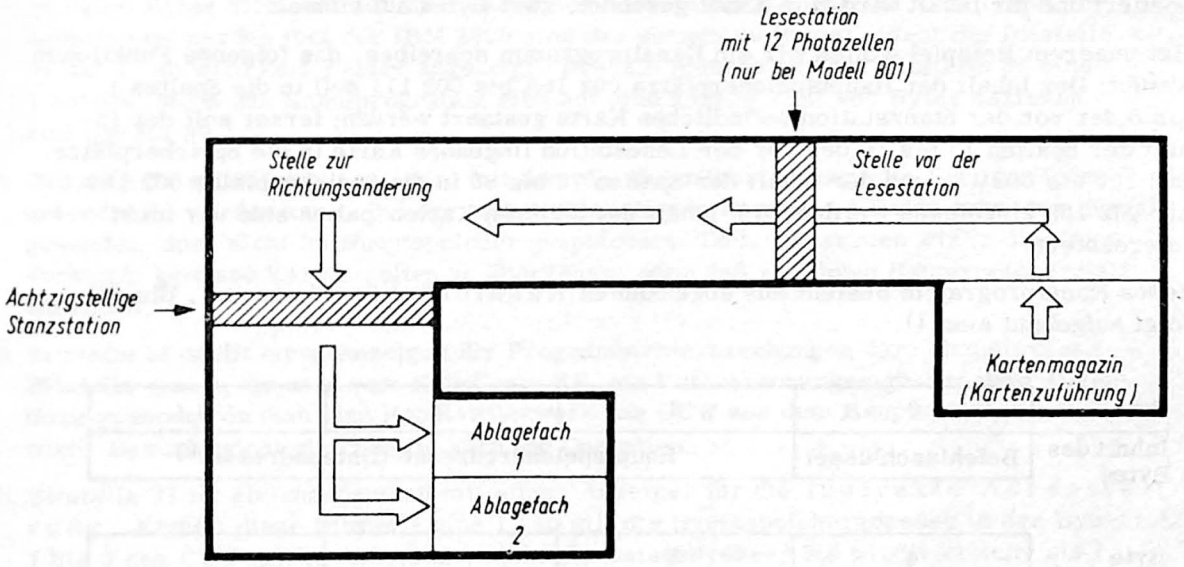


Abb. 5.2. Kartenbahn beim Lochkartengerät IBM 2520

Wenn eine Karte gelesen wird, so wird der Inhalt jeder Spalte in den EBCDI-Code übersetzt und jedes Byte wird in den Hauptspeicher übertragen. Der Leser selbst kann ein Byte speichern, so daß er mit einem Male zwei Bytes über den Kanal senden kann. Von der Leseoperation unterscheidet sich die Stanzoperation grundlegend. Vom Hauptspeicher her erreichen die Daten das Kartengerät seriell, d.h. Spalte auf Spalte, aber die Stanzeinrichtung arbeitet parallel, d.h. Zeile auf Zeile. Deshalb ist ein spezieller kleiner Kernspeicher eingebaut, der die Daten der 80 zu stanzenden Spalten speichert. Ein solcher Zwischenspeicher wird als Pufferspeicher oder kurz als Puffer bezeichnet. Wenn eine Karte gestanzt werden soll, so werden zunächst alle 80 Bytes vom Hauptspeicher in den Puffer übertragen. Erst nachdem der Puffer geladen ist, beginnt das Stanzen. Sobald sich eine Kartenzeile unter den Stanzstempeln befindet, durchsucht die IBM 2520 den Puffer, um zu bestimmen, welche Spalten in der betreffenden Zeile mit Lochungen versehen werden müssen. Nach dem Stanzen wird die Karte sofort in ein Ablagefach abgelegt.

Obgleich für die IBM 2520 eine ganze Reihe von Befehlsschlüsseln existiert, wollen wir für unser Beispiel nur drei betrachten:

C2	Lesen und Zuführen
09	Stanzen und Auswahl des Ablagefaches 1
01	Stanzen, Zuführen und Auswahl des Ablagefaches 1

Der Befehlsschlüssel 09 bewirkt, daß Daten vom Hauptspeicher zum Stanzpuffer übertragen werden und daß außerdem ein Schalter gesetzt wird, durch den erreicht wird, daß nach dem Stanzen die Karte ins Ablagefach 1 geht. (Zur Ablage der Karte im Ablagefach 2 gibt es andere Befehlsschlüssel). Jedoch wird keine Karte tatsächlich gestanzt, der Puffer wird bloß geladen, um das Stanzen vorzubereiten. Der Befehlsschlüssel 01 geht einen Schritt weiter als der Befehlsschlüssel 09. Er bewirkt, daß nach dem Laden des Puffers die Karte auch gestanzt wird: Die vor der Stanzstation liegende Karte wird gestanzt und die vor der Lesestation liegende Karte wird bis zur Stelle vor der Stanzstation weiterbewegt, ohne daß sie jedoch gelesen wird. Der Befehlsschlüssel C2 bewirkt, daß die vor der Stanzstation liegende Karte bis hinter die Stanzstation bewegt wird, und, wird dieser Befehlsschlüssel nach einem Befehlsschlüssel 09 gegeben, so wird die betreffende Karte außerdem gestanzt. Ferner wird die vor der Lesestation liegende Karte bis hinter die Photozellen transportiert und ihr Inhalt wird zum Kanal gesendet, zwei Bytes auf einmal.

Bei unserem Beispiel wollen wir ein Kanalprogramm schreiben, das folgende Funktionen ausübt: Der Inhalt der Hauptspeicherplätze 002 10A bis 002 111 soll in die Spalten 1 bis 8 der vor der Stanzstation befindlichen Karte gestanzt werden; ferner soll der Inhalt der Spalten 13 bis 16 der vor der Lesestation liegenden Karte in die Speicherplätze 002 100 bis 002 103 und der Inhalt der Spalten 75 bis 80 in die Speicherplätze 002 104 bis 002 109 eingelesen werden. Am Inhalt der anderen Kartenspalten sind wir nicht interessiert.

Jedes Kanalprogramm besteht aus sogenannten K a n a l b e f e h l s w ö r t e r n , die wie folgt aufgebaut sind [1]:

Byte	0	1	2	3
Inhalt des Bytes	Befehlsschlüssel	Hauptspeicheradresse (Datenadresse)[2]		

Byte	4	5	6	7
Inhalt des Bytes	Kennzeichen		Zähler	

Abb. 5.3. Der Aufbau der Kanalbefehlswörter (CCW)

Der Befehlsschlüssel (C2, 01 oder 09) belegt das erste Byte. Durch die nächsten drei Bytes ist die Hauptspeicheradresse festgelegt, bei der die Datenübertragung beginnt. Das 5. Byte (Byte 4) enthält verschiedene Kennzeichen. Das sechste Byte wird ignoriert, es kann deshalb beliebige Informationen enthalten. Durch die letzten beiden Bytes, das sogenannte Zählerfeld, wird die Anzahl Bytes festgehalten, die zu lesen bzw. die zu stanzen sind. Nach der Übertragung eines Bytes wird der Zähler um 1

1) Für Kanalfehlswörter ist die aus dem Englischen kommende Abkürzung CCW gebräuchlich (Channel Command Word).
2) Man beachte hierzu die Ausführungen über die indirekte Adressierung.

vermindert und die Hauptspeicheradresse um 1 erhöht. Die Datenübertragung wird dann beendet, wenn entweder der Zähler auf 0 abgebaut ist oder wenn das Ende der Karte erreicht ist.

a) Bitstelle 32 stellt den Anzeiger für Datenkettung (Chain Data Flag, kurz CD-Anzeiger) dar. Enthält diese Bitstelle eine 1, so wird eine Datenkettung ausgeführt. Das bedeutet folgendes: Wenn der Zähler 0 erreicht hat, so wird das nächste CCW herangezogen, um eine neue Hauptspeicheradresse, ein neues Kennzeichenbyte und einen neuen Zähler zu installieren. Der Befehlsschlüssel des neuen CCW wird dagegen vernachlässigt; es wird dieselbe Operation fortgeführt.

b) Bitstelle 33 stellt den Anzeiger für Befehlskettung (Chain Command Flag, kurz CC-Anzeiger) dar. Steht dieser Anzeiger auf 1 (ein), so wird, nachdem der Zähler auf 0 abgebaut ist, das nächste CCW einschließlich seines Befehlsschlüssels verwendet. Jedes CCW eines Kanalprogrammes mit Ausnahme des letzten CCW muß infolgedessen entweder auf Bitstelle 32 oder auf Bitstelle 33 eine 1 aufweisen.

c) Bitstelle 34 stellt den Anzeiger zur Unterdrückung einer falschen Satzlänge dar. Normalerweise wird ein Fehler angezeigt, wenn die Anzahl der im Kanalprogramm festgelegten Bytes nicht mit der Zahl der Bytes übereinstimmt, die von dem Gerät angesprochen werden (bei der IBM 2520 sind das gerade 80 Bytes). Steht die Bitstelle 34 auf 1, so wird kein Fehler angezeigt. Deshalb müssen wir auf Bitstelle 34 eine 1 setzen, wenn das Kanalprogramm sich auf eine andere Zahl von Bytes beziehen soll als auf 80.

d) Bitstelle 35 enthält den Anzeiger, mit dem die Datenübertragung zum Hauptspeicher unterdrückt werden kann. Bei derartigen Operationen werden die Daten zwar zum Kanal gesendet, doch nicht im Hauptspeicher gespeichert. Dadurch werden wir in die Lage versetzt, gewisse Kartenspalten zu übergehen, ohne daß wir ihnen Hauptspeicherplatz zuordnen.

e) Bitstelle 36 stellt einen Anzeiger für Programmunterbrechungen dar. Enthält diese Bitstelle eine 1, so wird vom Kanal zur ZE ein Unterbrechungssignal in dem Augenblick gesendet, in dem vom Kanalsteuerwerk das CCW aus dem Hauptspeicher geholt wird. Damit beginnt gleichzeitig auch die Operation.

f) Bitstelle 37 ist gleichbedeutend mit einem Anzeiger für die indirekte Adressierung. Enthält diese Bitstelle eine 1, so gilt die Hauptspeicheradresse in den Bytes 1 bis 3 des CCW (siehe Abb. 5.3.) nicht als Datenadresse. Sie wird vielmehr als Adresse von einem oder mehreren Wörtern angesehen, auf denen erst die eigentliche Datenadresse zu finden ist. Diese Wörter werden als indirekte Datenadreßwörter [1] (IDAW) bezeichnet; sie spezifizieren Hauptspeicheradressen für 1 bis einschließlich 2048 Bytes. Jedes IDAW muß eine durch 2048 teilbare Adresse beinhalten; eine Ausnahme hiervon macht nur das erste. Wenn der Anzeiger für die indirekte Adressierung gesetzt ist, so wird das erste IDAW durch die im CCW stehende Hauptspeicheradresse identifiziert. Wird diese Adresse um 4 erhöht, so ergibt sich die Lage des zweiten IDAW's im Hauptspeicher usw. Die Adresse, die im ersten IDAW gefunden wird, wird als Datenadresse vom CCW hergenommen und in üblicher Weise nach jedem übertragenen Byte um 1 erhöht. Diese Addition erfolgt solange, bis schließlich die ermittelte Hauptspeicheradresse durch 2048 teilbar ist. In diesem Moment wird das nächste IDAW herangezogen und in der gleichen Weise wie das erste benutzt.

1) Im angloamerikanischen Sprachraum als „indirect data address words", abgekürzt IDAW, bezeichnet.

g) Die Bitstellen 38 und 39 müssen Nullen enthalten. Eine 1 auf irgendeiner dieser beiden Stellen wird vom Kanal als Programmfehler angesehen.

Unser Kanalprogramm soll der Reihe nach folgende Operationen ausführen:

a) Zuerst soll der Inhalt der Hauptspeicherstellen 002 10A bis 002 111 in die ersten acht Stellen des Stanzpuffers übertragen werden (der Rest des Puffers soll leer bleiben).

b) Lesebefehle sollen folgen, mit deren Hilfe die ersten 12 Spalten übersprungen, die nächsten vier Spalten in die Hauptspeicherstellen 002 100 bis 002 103 eingelesen, die daran anschließenden 58 Spalten wieder übersprungen und die restlichen sechs Spalten auf die Hauptspeicherstellen 002 104 bis 002 109 gelesen werden.

Unser Kanalprogramm wird unter diesen Voraussetzungen folgendes Aussehen haben (siehe Abb. 5.4.).

Befehls-schlüssel	Hauptspei-cheradresse	Kennzei-chen	---	Zähler
09	002 10A	60	00	0008
C2	000 000	90	00	000C
00	002 100	80	00	0004
00	000 000	90	00	003A
00	002 104	00	00	0006

Abb. 5.4. Kanalprogramm für die IBM 2520 gemäß Aufgabenstellung

Durch das erste CCW wird eine Operation hervorgerufen, die das Stanzen und die Auswahl des Ablagefaches 1 bewirkt. Der CC-Anzeiger und der Anzeiger für falsche Satzlänge stehen hierbei auf 1. Das zweite CCW enthält einen Befehl zum Lesen und Zuführen, wobei der CD-Anzeiger und außerdem die Bitstelle 35 (Unterdrückung der Datenübertragung) auf 1 stehen. Aufgrund der Unterdrückung der Datenübertragung kann eine beliebige Hauptspeicheradresse in dieses CCW eingesetzt werden, wir haben 000 000 gewählt. Das dritte CCW beinhaltet eine Fortsetzung des Kartenlesens. Da der CD-Anzeiger des vorhergehenden CCW auf 1 steht, wird der Befehlsschlüssel dieses CCW ignoriert, wir haben deshalb den Befehlsschlüssel willkürlich auf 00 gesetzt. Da nur der CD-Anzeiger an ist, werden die vier angegebenen Kartenspalten auf die mit der Adresse 002 100 beginnenden vier Hauptspeicherbytes übertragen. Beim vierten CCW ist sowohl der CD-Anzeiger als auch der Anzeiger zur Unterdrückung der Datenübertragung auf ein gestellt; damit werden $003A_{16} = 58_{10}$ Kartenspalten übersprungen. Beim letzten CCW sind schließlich keine Anzeiger auf 1 gestellt; somit wird durch dieses CCW das Kanalprogramm beendet.

Das Schreiben eines Kanalprogrammes wie des obigen ist nicht allzu schwierig, doch erfordert es gründliche Kenntnisse über die Eigenschaften und die Arbeitsweise des vom Programm angesprochenen EA-Gerätes. Die hauptsächlich an die IBM /370 angeschlossenen Geräte wollen wir im Kap. 6 ausführlicher besprechen.

5.4. Die Verbindung zwischen Zentraleinheit und den Kanälen

In diesem Abschnitt wollen wir uns damit beschäftigen, wie die Zentraleinheit einem Kanal mitteilt, was er tun soll. Im darauffolgenden Abschnitt wollen wir schließlich die Frage abhandeln, wie die ZE vom Kanal erfährt, was er getan hat.

Durch EA-Operationen werden zwei feste Hauptspeicherplätze angesprochen. Das Kanaladreßwort (Channel Address Word, abgekürzt CAW) ist ein Vollwort und dem Speicherplatz 72_{10} fest zugeordnet. Es besitzt folgenden Aufbau:

Byte	0								1	2	3
Inhalt des Bytes	Schlüssel				0	0	0	0	Hauptspeicheradresse (Befehlsadresse)		
Bitstellen	0	bis	3		4	5	6	7	8	bis	31

Abb. 5.5. Der Aufbau des Kanaladreßwortes (CAW)

Durch die erste Sedezimalziffer ist der Speicherschutzschlüssel festgelegt. Die zweite Sedezimalziffer muß immer gleich 0 sein. Die letzten drei Bytes geben die Hauptspeicheradresse an, auf der das erste CCW des Kanalprogrammes zu finden ist.

Das Kanalstatuswort (Channel Status Word, abgekürzt CSW) ist ein Doppelwort und dem Speicherplatz 64_{10} fest zugeordnet. Das CSW besitzt folgenden Aufbau:

Byte	0								1	2	3
Inhalt des Bytes	Schlüssel				0	A	0	0	Hauptspeicheradresse (Befehlsadresse)		
Bitstellen	0	bis	3		4	5	6	7	8	bis	31

Byte	4			5			6			7	
Inhalt des Bytes	Gerätestatus			Kanalstatus			Zähler				
Bitstellen	32	bis	39	40	bis	47	48	bis			63

Abb. 5.6. Der Aufbau des Kanalstatuswortes (CSW)

Nachstehend wollen wir zunächst in kurzen Zügen Funktion und Inhalt des Kanalstatuswortes beschreiben. Das Kanalstatuswort (CSW) wird normalerweise nach Beendigung einer EA-Operation auf seinem festen Hauptspeicherplatz gespeichert, aber es kann unter gewissen Umständen auch schon während des Ablaufes einer Operation auf diesen abgelegt werden. Der Schlüssel wird vom Kanaladreßwort (CAW) übernommen, mit dem das Kanalprogramm gestartet wurde. Die Bytes 1 bis 3 des CSW enthalten die Hauptspeicheradresse des nächsten zu benutzenden Kanalbefehlswortes (CCW); diese ist normalerweise gleich der um 8 vermehrten Adresse des augenblicklich benutzten Kanalbefehlswortes. Die vierte Stelle des Kanalstatuswortes nimmt das Gerätestatusbyte auf, die fünfte das Kanalstatusbyte. Die letzten beiden Bytes schließlich enthalten den verbleibenden Wert des Zählers des zuletzt benutzten CCW. Wird das CCW am Schluß einer EA-Operation gespeichert, so ist der Zählerstand normalerweise gleich Null.

Ein paar Anmerkungen zu den restlichen, noch nicht besprochenen Bitstellen des Bytes 0 mögen jetzt folgen. Die Bitstellen 6 und 7 enthalten den Bedingungsschlüssel. In wenigen Fällen wird das der Bedingungsschlüssel sein, der von der Instruktion gesetzt wurde, durch die das Kanalprogramm gestartet wurde. In den meisten Fällen werden auf diesen beiden Bitstellen jedoch Nullen auftreten. Auf der Bitstelle 4 muß eine Null vorhanden sein. Auch Bitstelle 5 ist gewöhnlich mit einer Null versehen. Findet sich auf Bitstelle 5, in Abb. 5.6. durch A gekennzeichnet, jedoch eine 1, so bedeutet das, daß eine technische Unzulänglichkeit oder Unstimmigkeit im Kanal vorliegt, wodurch die Ausführung des Kanalprogrammes verhindert wurde. Über die Art des aufgetretenen Problems wird das Wartungspersonal durch eine aus mehreren Bitstellen bestehende Anzeigeleiste im Kanal[1] informiert. Abhängig von den Bits im Steuerregister 14 kann die Anzeigeleiste aufgenommen werden. Damit ist gemeint, daß der Kanal in feste Hauptspeicherstellen im niedrigen Adreßbereich die Bits aufzeichnet, die die betreffende Bedingung beschreiben. Danach versucht der Kanal, sich selbst wieder in den Normalzustand zu versetzen, so daß er wieder angesprochen und benutzt werden kann. Da viele elektronische Probleme in frühen Stadien vorübergehender Natur sind, besteht eine gute Chance, daß das Wiederaufsetzen von Kanälen auch zum Erfolg führt. Gelingt dies jedoch nicht, so hat sich für den betreffenden Kanal ein inoperativer Zustand eingestellt.

Die beiden Status- oder Zustandsbytes enthalten Informationen, die den Zustand des EA-Gerätes und des Kanals beschreiben. Dazu zählen z.B. folgende Hinweise: a) Ungültiger Befehlsschlüssel in einem CCW, b) Datenfehler aufgrund der Paritätsprüfung, c) EA-Gerät bereits beschäftigt u.a.m.

Es gibt acht verschiedene EA-Instruktionen. Alle acht gehören der Klasse der privilegierten Instruktionen an und können infolgedessen nur im Supervisorstatus (siehe dazu Abschnitt 4.11.) ausgeführt werden. Für die EA-Instruktionen gilt hauptsächlich das sogenannte S-Format, in Abb. 5.7. ist es dargestellt.

Byte	0	1	2	3
Inhalt des Bytes	Operationsschlüssel		B_2	D_2

Abb. 5.7. Instruktion im S-Format

Da diese Instruktionen nur zu Speicherplätzen und nicht zu Registern zugreifen, hat man für dieses Format den Kennbuchstaben S (von Speicher) festgelegt. Interessant an diesem Format ist die Tatsache, daß für den Operationsschlüssel die ersten beiden Bytes und nicht nur das erste Byte wie bei den anderen Formaten reserviert sind. Viele der Instruktionen im S-Format benötigen dennoch nur das erste Byte für den Operationsschlüssel; das zweite Byte kann in diesen Fällen völlig frei benutzt werden. Bei anderen wiederum ist zwar der Operationsschlüssel auf das Byte 1 expandiert, belegt dort jedoch nur die letzte Bitstelle dieses Bytes. Wegen dieser Unterschiede müssen wir bei allen Instruktionen, die im S-Format vorliegen, genau umreißen, welchen Inhalt das zweite Byte aufweisen muß.

Bei den EA-Instruktionen wird der Inhalt der Speicherplätze mit der effektiven Adresse $E_2=(B_2)+D_2$ so interpretiert, wie es die Abb. 5.8. zeigt.

1) Im angloamerikanischen Sprachraum als „Channel Logout" bezeichnet.

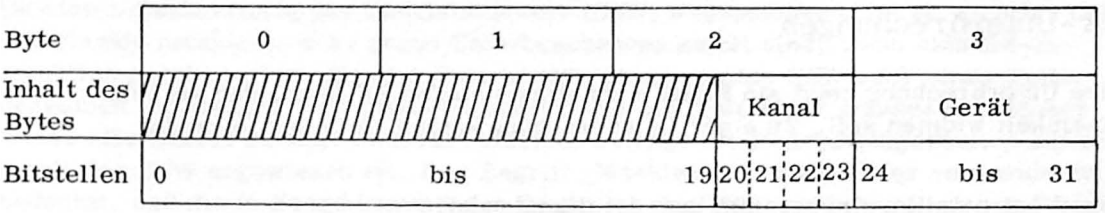


Abb. 5.8. Effektive Adresse bei EA-Instruktionen

Der Operationsschlüssel der EA-Instruktionen setzt sich aus dem Inhalt der Bitstellen 0 bis 7 und der Bitstelle 15 zusammen. Die Bitstellen 8 bis 14 werden nicht benutzt und können infolgedessen einen beliebigen Inhalt aufweisen. In herkömmlicher Weise werden deshalb diese Bitstellen mit Nullen versehen. Es ist zu beachten, daß die von den EA-Instruktionen gebildete Adresse nicht die Adresse eines Hauptspeicherplatzes darstellt. Durch sie wird vielmehr die Adresse eines EA-Gerätes repräsentiert. Das rechtsbündige Byte dieser Adresse (siehe Abb. 5.8.) gibt die Geräteadresse selbst an, während die links davor stehenden vier Bitstellen sich auf den Kanal beziehen. Der Inhalt der anderen Bitstellen wird ignoriert.

Um den Ablauf eines Kanalprogrammes einzuleiten, muß eine geeignete Instruktion, z.B. SIO [1] gegeben werden. Ihre Ausführung veranlaßt der Reihe nach folgende Schritte:

1) Die EA-Adresse wird entschlüsselt und der Kanal geprüft. Ist der Kanal nicht arbeitsfähig (z.B. wenn er nicht unter Spannung steht), so wird der Bedingungsschlüssel auf 3 gesetzt und die Ausführung der Instruktion wird beendet.

2) Arbeitet der Kanal im Einpunktbetrieb, so wird der Bedingungsschlüssel auf 2 gesetzt und die Ausführung der Instruktion wird beendet.

3) Die Statusbytes für den Kanal und das Gerät werden geprüft. Zeigt eines dieser beiden einen Grund auf, der verhindert, daß die EA-Operation stattfinden kann, so werden die beiden Statusbytes in das CSW gespeichert (der Rest des CSW bleibt unberührt). Die Ausführung der Instruktion wird beendet, nachdem zuvor noch der Bedingungsschlüssel auf 1 gesetzt wurde.

4) Ist keine Sonderbedingung entdeckt worden, so müssen Kanal und EA-Gerät für die Ausführung einer EA-Operation verfügbar sein: Der Bedingungsschlüssel wird auf 0 gesetzt, das CAW wird in den Kanal geholt, der Kanal beginnt zu arbeiten und die ZE geht zur nächsten Instruktion über.

Zu diesem Zeitpunkt können wir statuieren, daß wir nur in groben Umrissen wissen, wie eine EA-Operation gestartet werden kann. Man stellt hierzu zunächst die Adresse des ersten Kanalbefehlswortes (CCW) in das Kanaladreßwort (CAW), setzt eine geeignete Instruktion auf und überprüft anschließend mittels einer BC-Instruktion den Bedingungsschlüssel. Wird dabei dieser zu ungleich Null gefunden, so verzweigt man sinnvollerweise zu einer Routine, die analysiert, warum die aufgesetzte Operation nicht ausgeführt werden konnte.

1) SIO ist der mnemonische Operationsschlüssel der Instruktion „Starten Eingabe/Ausgabe", im angloamerikanischen Sprachraum „Start Input/Output".

5. 5. EA-Unterbrechungen

Durch eine Unterbrechung zeigt ein Kanal normalerweise der ZE [1] an, daß sie ihm Aufmerksamkeit widmen soll. Zu einer Unterbrechung kommt es in den folgenden Fällen:

a) Ein Kanal hat sein Kanalprogramm beendet (Kanalende).

b) Ein EA-Gerät hat die ihr aufgegebene Operation beendet (Geräteende).

c) Bei einem EA-Gerät ist ein Sonderzustand eingetreten.

d) Von einem Kanal wird ein Kanalbefehlswort (CCW) gelesen, dessen Bitstelle 36 eine 1 enthält.

Tritt eine durch die EA bedingte Unterbrechung auf, so wird das PSW (Programmstatuswort) des gerade ablaufenden Programmes auf den Speicherplatz 56 weggespeichert. Bei Vorliegen der Basissteuerung (siehe Abschnitt 4.11.) wird in die rechtsbündigen 12 Bitstellen des Unterbrechungsschlüssels im PSW, d.h. also in die Bitstellen 20 bis 31 (siehe Abb. 4.13.), die Kanalgeräteadresse gebracht, die davor gelegenen restlichen vier Bitstellen des Unterbrechungsschlüssels werden mit Nullen aufgefüllt. Bei der erweiterten Steuerung hingegen enthalten die Bytes mit den Hauptspeicheradressen 185 bis 187 in sedezimaler Form 000kgg, wobei k die Kanaladresse und g die Geräteadresse darstellen; somit enthalten die Bytes mit den Adressen 186 und 187 die gleichen Angaben wie der Unterbrechungsschlüssel im PSW bei der Basissteuerung. Nach dem Speichern des alten Programmstatuswortes wird das neue Programmstatuswort vom Speicherplatz 120 geladen. Damit kann die Bearbeitung der Unterbrechung eingeleitet werden. Eine EA-Unterbrechung veranlaßt weiterhin die Speicherung des gesamten Kanalstatuswortes (CSW) auf den Speicherplatz 64. Alle Adreßangaben der soeben gemachten Ausführungen beziehen sich auf dezimale Adressen (siehe auch Abb. 4.12.).

Bei bestimmten Voraussetzungen ist es für eine ZE nicht wünschenswert, daß sie sich unterbrechen läßt. Im erweiterten Steuerungsmodus sieht das so aus: Enthält die Bitstelle 6 des PSW eine 1, so sind EA-Unterbrechungen bei jedem Kanal möglich, für den die entsprechende Bitposition im Steuerregister 2 eine 1 enthält. Eine 0 auf der Bitstelle 6 der PSW hingegen bewirkt, daß keine EA-Unterbrechungen durchkommen und alle hängenden Unterbrechungen im Kanal hängenbleiben. Eine hängende Unterbrechung führt erst dann zur Unterbrechung, wenn ein PSW geladen ist, das auf Bitstelle 6 keine 0 aufweist oder wenn der Inhalt des Steuerregisters 2 so modifiziert ist, daß eine Unterbrechung möglich ist; natürlich muß dabei auf Bitstelle 6 des PSW eine 1 vorhanden sein. Sind diese Voraussetzungen erfüllt, so tritt die Unterbrechung so bald als möglich ein. Bei der Basissteuerung werden die Bitstellen 0 bis 5 des Steuerregisters 2, die den Kanälen 0 bis 5 bei der erweiterten Steuerung entsprechen, nicht benutzt. An deren Stelle treten die Bitstellen 0 bis 5 des PSW selbst, die in gleicher Weise die Unterbrechungen regeln. Die Bitstelle 6 des PSW und das Steuerregister 2 funktionieren wie bei der erweiterten Steuerung für die Kanäle 6 bis 14. Die Bitstellen 15 bis 31 des Steuerregisters 2 werden bei keinem Steuerungsmodus benutzt.

Hat eine Unterbrechung stattgefunden, so verbleibt sie nicht länger im Hängezustand und wir sagen deshalb, daß sie geklärt ist.

---

1) ZE ist die Abkürzung von Zentraleinheit.

Da eine Unterbrechung das Kanalstatuswort (CSW) wegspeichert, ist es wichtig, daß alle Kanäle maskiert, d.h. gegen Unterbrechungen gefeit sind, wenn eine EA-Instruktion gegeben wird, die eine Speicherung des gesamten CSW oder eines Teils desselben verursacht. Wird eine Unterbrechung bearbeitet, so müssen in gleicher Weise alle Kanäle solange maskiert bleiben, solange die Bearbeitungsroutine auf den Inhalt des CSW angewiesen ist. Der Begriff „Maskieren", wie er hier verwendet wird, bedeutet, daß die in Frage kommenden Bitstellen des PSW und bzw. oder des Steuerregisters so gesetzt sind, daß keine Unterbrechung stattfinden kann.

Zu diesem Zeitpunkt wollen wir einen ersten Blick auf den normalen Programmablauf für die EA bei der IBM /370 werfen:

1. Im Programm für die ZE wird eine SIO-Instruktion [1] gegeben, die den Bedingungsschlüssel 0 zur Folge hat.
2. Die EA-Maskenbitstellen sind so gesetzt, daß sie eine Unterbrechung seitens des betreffenden Kanals gestatten.
3. Die ZE setzt die Verarbeitung fort, während der Kanal sein eigenes Programm ausführt.
4. Am Ende des Kanalprogrammes erzeugt der Kanal eine Unterbrechung. Die Behandlungsroutine für diese Unterbrechung prüft zunächst, ob die EA-Operation normal beendet wurde. Trat hingegen während der EA ein Fehler auf, so wird die geeignete Maßnahme veranlaßt. Um die normale Beendigung einer EA-Operation zu signalisieren, wird eine bestimmte Bitstelle durch eine 1 gekennzeichnet.
5. Erreicht das Anwendungsprogramm den Punkt, an dem es nicht mehr weiterlaufen kann, ohne beispielsweise auf die durch die initiierte EA-Operation gelesenen Informationen zurückgreifen zu können, so wird zunächst die unter 4. erwähnte Bitstelle auf ihren Inhalt überprüft. Findet das Anwendungsprogramm auf dieser Bitstelle eine 1 vor, so nimmt es die Daten zur Verarbeitung her und überträgt sie evtl. in ein Arbeitsgebiet. Anschließend wird die nächste EA-Operation eingeleitet.
6. Enthält die Kennzeichenbitstelle (siehe Pkt. 4.) dagegen eine Information, die aussagt, daß die betreffende EA-Operation noch nicht vollendet ist, so geht das Programm in den Wartezustand über. Die Behandlungsroutine für EA-Unterbrechungen setzt die Wartebitstelle im PSW (Bitstelle 14, siehe Abb. 4.13.) wieder auf 0 zurück, bevor das alte PSW erneut geladen wird.

In der Praxis wird das Hauptprogramm (Anwendungsprogramm) jedoch allein für den Problemstatus geschrieben, so daß es niemals eigene EA-Operationen ausführen kann. Das bedeutet aber, daß ein Supervisorprogramm vorhanden sein muß, das diese Funktionen vom Anwendungsprogramm übernimmt und ausführt. Ein solches Supervisorprogramm kann tatsächlich für die meisten Anwendungsprogramme gleich sein. Infolgedessen braucht der Programmierer nicht in jedem Fall sein eigenes Supervisorprogramm zu schreiben. Die Verbindung zwischen dem Problemprogramm (Anwendungsprogramm für den Problemstatus) und dem Supervisorprogramm wird gewöhnlich mittels der Instruktion SVC [1] vorgenommen, über die wir im Kapitel 7 sprechen wollen.

---

1) SIO ist der mnemonische Operationsschlüssel der Instruktion „Starten Eingabe/Ausgabe", englisch „Start Input/Output".

1) Supervisoraufruf; englisch „Supervisor Call", abgekürzt SVC.

Einige recht sorgfältig durchdachte Supervisorprogramme sind von der IBM geschrieben worden und stehen allen Benutzern des Systems IBM /370 zur Verfügung. Einige dieser Programme wollen wir in der zweiten Hälfte dieses Buches diskutieren. Die richtige Verwendung dieser Programme hängt in einem gewissen Grade davon ab, ob man einen Überblick über die Methode gewonnen hat, in welcher Weise einerseits die ZE mit den Kanälen und andererseits die Problemprogramme mit den Supervisorprogrammen zusammenspielen.

5.6. Statusbytes und Prüfbytes

Jedes EA-Gerät besitzt ein Statusbyte, das als Teil des Kanalstatuswortes (CSW) gespeichert wird. In diesem Byte wird der jeweilige Zustand des EA-Gerätes festgehalten. Die einzelnen Bitstellen dieses Statusbytes sollen fortlaufend mit U_0 bis einschließlich U_7 bezeichnet werden. Sie haben folgende Bedeutung:

1) U_0 — Achtung
Diese Bitstelle wird dann auf 1 gesetzt, wenn die Maschinenbedienung die dazugehörige Taste an dem betreffenden Gerät gedrückt hat.

2) U_1 — Statusänderung
Diese Bitstelle wird in Verbindung mit U_3 und U_5 dazu benutzt, um den normalen Status des Gerätes zu ändern.

3) U_2 — Steuereinheitenende
Diese Bitstelle wird auf 1 gesetzt, wenn die Steuereinheit ihren Anteil an der EA-Operation beendet hat. Dieses Signal wird gewöhnlich unterdrückt; es sei denn, daß eine einzige Steuereinheit für mehrere EA-Geräte vorhanden ist, und daß diese getestet und dabei festgestellt wurde, daß sie sich in Arbeit befindet. Sie wird sofort auf 1 gebracht, sobald die Steuereinheit frei wird.

4) U_3 — In Arbeit
Ist diese Bitstelle auf 1 gesetzt, so wird damit angezeigt, daß das Gerät sich in Arbeit befindet, indem es eine vorhergehende Operation ausführt. Wird gleichzeitig auch U_1 auf 1 gesetzt, so heißt das, daß die Steuereinheit und nicht das EA-Gerät beschäftigt ist.

5) U_4 — Kanalende
Der Unterkanal hat seinen Anteil an der EA-Operation, nämlich die Datenübertragung, erledigt. Das Gerät selbst kann seinen Arbeitsanteil ebenfalls bereits vollendet haben, es muß ihn aber noch nicht beendet haben.

6) U_5 — Geräteende
Das EA-Gerät hat seine Funktion erfüllt und steht für eine andere Operation bereit. Geräteende wird bei den EA-Geräten auch dann signalisiert, wenn ein zuvor stillgelegtes Gerät in Betrieb genommen wird. Wird neben U_5 auch U_1 auf 1 gesetzt, so bedeutet das, daß eine EA-Operation außergewöhnlich, doch gültig, zu Ende gegangen ist.

7) U_6 — Gerätefehler
Erscheint auf dieser Bitstelle eine 1, so bedeutet das im Normalfall einfach, daß mindestens eine Bitstelle des Prüfbytes (bzw. der Prüfbytes) des EA-Gerätes auf 1 steht. Im Programm sollte dann untersucht werden, welche Prüfstelle auf 1 gesetzt wurde. Dadurch kann festgestellt werden, welche Bedingung am betreffenden Gerät vorliegt.

8) U_7 — Geräte-Ausnahmebedingung
Es ist eine Ausnahmebedingung, die jedoch gültig ist, entstanden. Das bedeutet z.B. beim Drucker, daß das Ende einer Seite erreicht wurde, beim Kartenleser, daß die letzte Karte gelesen wurde und beim Magnetband, daß eine Bandrolle zu Ende gegangen ist.

Jedes EA-Gerät besitzt neben dem Statusbyte auch ein oder mehrere Prüfbytes. Die Prüfbytes sind verschieden aufgebaut; die Bedeutung ihrer Bitstellen hängt von den einzelnen Geräten ab. Nur die ersten sechs Bitstellen haben für alle EA-Geräte die gleiche Bedeutung:

1) Bitstelle 0 — Zurückweisung eines Befehls
Das EA-Gerät kann den gegebenen Kanalbefehl nicht ausführen.

2) Bitstelle 1 — Eingriff erforderlich
Es ist ein manueller Bedienereingriff erforderlich. Der Grund ist darin zu suchen, daß bei Kartengeräten beispielsweise ein Kartenanstoß erfolgte oder daß beim Drukker das Papier ausgegangen ist usw.

3) Bitstelle 2 — Fehler im Ausgangsübertragungsweg
Das EA-Gerät hat vom Unterkanal ein Zeichen mit einer geraden Anzahl von Einerbits empfangen.

4) Bitstelle 3 — Ausrüstungsprüfung
Es tritt irgendein Fehler in der Ausrüstung des EA-Gerätes auf.

5) Bitstelle 4 — Datenprüfung
Es wurden ungültige Daten (z.B. nichtexistierende Lochkombinationen bei Lochkarten) entdeckt. Eine 1 auf dieser Bitstelle besagt, daß irgendein Fehler beim externen Medium (beim Datenträger) vorliegt und nicht beim EA-Gerät.

6) Bitstelle 5 — Datenverlust
Dem Kanal ist es nicht gelungen, innerhalb des zugelassenen Zeitintervalles auf die Anforderung eines EA-Gerätes, es zu bedienen, zu reagieren. Das heißt aber, daß die gesamte Übertragungsgeschwindigkeit aller in Gang befindlichen EA-Operationen größer ist als die Verarbeitungsgeschwindigkeit im Hauptspeicher des Computers.

Die genaue Bedeutung jeder einzelnen Bitstelle der Prüfbytes ist für jedes EA-Gerät gesondert definiert. Ebenso hängt die Anzahl der vorhandenen Prüfbytes von den einzelnen Geräten ab. Die Prüfbytes können wie Daten in den Hauptspeicher eingelesen werden, wenn ein Lesebefehl gegeben wird, der speziell nur für die Prüfbytes gilt. Die Prüfbytes werden zu Beginn einer Lese- bzw. Schreiboperation gelöscht, d.h. auf Null gesetzt. Hiervon gibt es einige Ausnahmen. Z.B. bleibt der Zustand „Bedienereingriff erforderlich" (Bitstelle 1) so lange bestehen, solange dieser Zustand andauert.

Ist einmal eine Bitstelle des Statusbytes auf 1 gesetzt worden, so bleibt ihr Wert solange erhalten, solange das Statusbyte nicht neu gesetzt wird. Das Statusbyte wird stets nach dem Speichern seines Inhaltes ins Kanalstatuswort (CSW) neu gesetzt.

5.7. Einleitendes Programmladen[1)]

Beim erstmaligen Einschalten eines Computers müssen einige Maßnahmen ergriffen werden, um ein Programm in den Hauptspeicher zu laden und die Ausführung dieses Programmes zu starten. Bei der IBM /370 wird das einleitende Programmladen wie folgt durchgeführt: Mittels dreier am Steuerpult (Konsol) befindlicher Drehschalter wird eine Kanal-Geräte-Adresse eingestellt und die Ladetaste (IPL-Taste) gedrückt. Bei einigen Modellen der Computerfamilie IBM /370 sind jedoch keine Drehschalter vorhanden. Bei diesen Modellen ist die Modusauswahltaste (Modus Select) zu drücken und danach der Buchstabe L gefolgt von der Kanal-Geräte-Adresse einzutasten. Danach ist die Taste „Eingabe" (Enter) zu drücken, um das einleitende Programmladen zu veranlassen. Das einleitende Programmladen bewirkt der Reihe nach folgendes:

1) Der Computer, seine Kanäle und seine EA-Geräte werden gelöscht, das bedeutet, daß alle Anzeiger, Statusbytes und Prüfbytes auf 0 gesetzt werden. Jede in Gang befindliche EA-Operation wird abgestoppt.

2) Der Computer erzeugt intern die Ausführung einer SIO-Instruktion mit dem Kanaladreßwort (CAW) 0000 0000 und dem Kanalbefehlswort (CCW) 02 000000 60 000018, wobei die durch die Konsoldrehschalter eingestellte bzw. die eingegebene Kanal-Geräte-Adresse verwendet wird. Durch den Befehlsschlüssel 02 ist ein Standardlesebefehl gekennzeichnet, der von allen Eingabegeräten verstanden und ausgeführt werden kann. Nachdem die ersten 24 Bytes (18_{16} Bytes) in die ersten 24 Hauptspeicherplätze eingelesen sind, nimmt der Kanal das nächste Kanalbefehlswort (CCW) vom Hauptspeicherplatz 8 her, da der Anzeiger für die Befehlskettung auf 1 steht (siehe hierzu Abschnitt 5.3., Abb. 5.3.).

3) Ist das Kanalprogramm abgeschlossen, so geht der Computer automatisch in den Operationsstatus über, wobei er als PSW das Doppelwort auf dem Speicherplatz 0 verwendet. Das Doppelwort auf Speicherplatz 8 wird als erstes CCW benutzt (CCW1), das Doppelwort auf Speicherplatz 16 als zweites CCW (CCW2).

Die Wirkungsweise von IPL sei noch einmal zusammengefaßt: Jede in Gang befindliche EA-Operation wird angehalten, vielerlei wird gelöscht, und es werden außerdem ein PSW und zwei CCW eingelesen. Diese CCW werden dann dazu benutzt, um den Rest des Programmes zu holen. Danach wird die Ablaufsteuerung des Computers dem gerade gespeicherten PSW übergeben.

Da der Anzeiger zur Unterdrückung einer falschen Satzlänge genau so wie der Anzeiger für Befehlskettung an ist, werden nur die ersten 24 Bytes des ersten Satzes eingelesen. Wird zum Programmladen eine Karte verwendet, so heißt das also, daß nur die ersten 24 Kartenspalten übertragen werden.

---

1) Im angloamerikanischen Sprachraum als „Initial Program Loading', abgekürzt IPL bezeichnet.

Aufgabengruppe 11

1) Jedes der folgenden Kanalprogramme wurde für die IBM 2520 entworfen. Ermittle die Fehler!

   (a) C2 001000 00 000000

   (b) C2 001000 00 000080

   (c) C2 001000 02 000050

   (d) 0 1 001000 40 000050

   C2 001000 30 000000

2) Schreibe die nötigen Kanalbefehlswörter, die folgende Aufgabenstellungen lösen:

   a) Einlesen der Kartenspalten 1 bis 50 und Speicherung ihres Inhaltes auf die Speicherplätze ab 001 000; Einlesen der Kartenspalten 51 bis 80 und Speicherung ihres Inhaltes auf die Speicherplätze ab 001 283.

   b) Stanzen des Inhaltes der Speicherplätze 001 000 bis 001 03E auf die Kartenspalten 1 bis 63; die Kartenspalten 64 bis 80 sollen leer bleiben.

   c) Stanzen des Inhaltes der Speicherstellen 001 000 bis 001 03E auf die Kartenspalten 1 bis 63; Lesen der Kartenspalten 64 bis 80 der nächsten Karte in die Speicherplätze von 001 03F bis 001 04F.

   d) Lesen zweier Karten. Der Inhalt der ersten Karte soll auf die Speicherplätze 001 000 bis 001 04F und der Inhalt der zweiten Karte auf die Speicherplätze 001 050 bis 001 09F übertragen werden.

3) Beim Versuch, das für eine IBM 2520 geschriebene Kanalprogramm

   C2 001000 00 000050

   nach einer SIO-Instruktion ausführen zu lassen, hatte zur Folge, daß der Bedingungsschlüssel auf 1 gesetzt und keine Karte eingelesen wurde. Eine Überprüfung der Speicherplätze 000 044 und 000 045 zeigte, daß diese Bytes den Inhalt 01 00 aufwiesen. Welche Bedingung war eingetreten? Nimm den Basissteuerungsmodus an!

4) Es liegt derselbe Vorgang wie bei Aufgabe 3) vor, nur weisen die Speicherplätze 000 044 und 000 045 den Inhalt 10 00 auf. Wie kann dieser Vorgang erklärt werden?

5) Das Kanalprogramm

   01 00 1000 40 000050
   C2 00 1000 28 000020

   ist für eine IBM 2520 durch eine SIO-Instruktion gestartet worden. Der Bedingungsschlüssel ergab sich zu 0. Es trat eine Unterbrechung auf und eine anschließende Überprüfung der Speicherplätze 000 044 und 000 045 ergab, daß ihr Inhalt 00 80 war. Wodurch kam die Unterbrechung zustande?

5.8. Das Kanalbefehlswort (CCW)

Bis jetzt haben wir die Eingabe/Ausgabeoperationen bei der IBM /370 nur allgemein diskutiert. In den restlichen Abschnitten dieses Kapitels wollen wir das bisher Besprochene erneut abhandeln und dabei mehr Einzelheiten kennenlernen.

Kanalbefehlsschlüssel können in folgende Gruppen eingeteilt werden:

Schlüssel	Operation
xxxx 0000	(ungültig)
xxxx 1000	Verzweigen im Kanal (Transfer im Kanal[1])
yyyy yy01	Schreiben
yyyy yy11	Steuerung
yyyy yy10	Lesen
yyyy 0100	Prüfen
yyyy 1100	Lesen rückwärts

Durch die Buchstaben x soll angezeigt werden, daß die betreffenden Bitstellen für die Durchführung der Operation ohne Bedeutung sind; durch die Buchstaben y soll dagegen angezeigt werden, daß die betreffenden Bitstellen eine bestimmte Bedeutung besitzen; sie sind maßgebend dafür, in welcher Weise die entsprechende Grundoperation modifiziert wird (Ablagefach beim Stanzer usw.).

Alle Befehlsschlüssel, deren Bitstellen 4 bis 7 Nullen enthalten, sind ungültig.

Der Befehlsschlüssel für den Befehl „Verzweigen im Kanal" kann innerhalb eines Kanalprogrammes an beliebiger Stelle erscheinen. Das CCW mit diesem Befehlsschlüssel bleibt bis auf das Adreßfeld unbeachtet. Dieses gibt die Adresse des Speicherplatzes an, auf dem das nächste CCW des Kanalprogrammes zu finden ist. Somit läuft ein solcher Befehl auf eine unbedingte Verzweigung innerhalb des Kanalprogrammes hinaus. Es ist jedoch nicht erlaubt, daß das erste CCW oder zwei aufeinanderfolgende CCW eines Kanalprogrammes diesen Befehlsschlüssel aufweisen.

Mit Ausnahme der ungültigen Befehlsschlüssel und des Befehlsschlüssels für den Befehl „Verzweigen im Kanal" bedeutet jeder Befehlsschlüssel mit einer 1 in der Bitstelle niedrigster Ordnung einen Schreibbefehl und jeder Befehlsschlüssel mit einer 0 einen Lesebefehl. Die Steuerung kann grundsätzlich als spezieller Schreibbefehl aufgefaßt werden; sie bewirkt, daß ein Signal zum EA-Gerät gesendet wird, jedoch daß keine Daten auf das externe Speichermedium (auf den Datenträger) geschrieben werden. Durch die Steuerungsbefehle werden solche Vorgänge eingeleitet, wie der Papiertransport beim Drucker, wie das Zurückspulen von Magnetbändern usw. Ein spezieller Fall eines Steuerungsbefehls ist der folgende:

0000 0011

Hierdurch wird das EA-Gerät veranlaßt, keine mechanische Funktion auszuführen, jedoch unmittelbar nach der Entschlüsselung dieses Befehles Kanalende und Geräteende zu generieren.

Das Prüfen ist ein spezieller Fall eines Lesebefehls; durch ihn werden nämlich die Prüfbytes in den Hauptspeicher eingelesen, jedoch keine Daten. Auch beim Befehl „Lesen rückwärts" handelt es sich um einen speziellen Lesebefehl. Bei gewissen Geräten veranlaßt dieser Befehl, daß die Daten in Hauptspeicherplätze mit fallenden Adressen eingelesen werden, d.h. rückwärts. Dadurch ist es möglich, daß Magnetbänder in beiden Richtungen gelesen werden können.

Zu einer Datenkettung kommt es, wenn das Zählerfeld auf 0 aufgebaut ist und der CD-Anzeiger (Bitstelle 32) auf 1 steht. Die Datenkettung wird unterdrückt und die Operation beendet, wenn die Bitstellen 1 (falsche Länge), 2 (Programmfehler), 3 (Speicherschutzfehler), 6 (Kanalsteuerfehler), 6 (Interface-Steuerfehler) oder 7 (Kettungsfeh-

1) Im angloamerikanischen Sprachraum als „Transfer in Channel", abgekürzt TIC bezeichnet.

ler) des Kanalstatusbytes[1] eine 1 enthalten. Die Datenkettung braucht nicht unterdrückt zu werden, wenn die Bitstelle 4 (Kanaldatenfehler) eine 1 enthält. Durch die Datenkettung ist es möglich, daß die zu einem physikalischen Satz gehörenden Informationen über mehrere Hauptspeichergebiete verteilt werden. Die Datenkettung stellt jedoch kein Mittel dar, um mehrere physikalische Sätze auf externen Medien anzusprechen.

Im Grunde erlaubt die Befehlskettung, daß der Kanal automatisch eine andere EA-Operation startet, z.B. den nächsten physikalischen Satz heranzieht, wenn die vorhergehende Operation abgeschlossen ist. Voraussetzung hierfür ist, daß dieser Abschluß auf normalem Wege erfolgt. Befehlskettung betrifft immer Informationen von verschiedenen physikalischen Sätzen. Es ist nicht möglich, mittels einer Befehlskettung verschiedene Teile eines einzigen physikalischen Satzes anzusprechen. Befehlskettung liegt dann vor, wenn die Bitstelle 32 eine 0 und die Bitstelle 33 (CC-Anzeiger) eine 1 enthält. Am Ende einer Operation wartet der Kanal darauf, daß Kanalende und Geräteende signalisiert werden, um anschließend sofort die Statusbytes zu prüfen. Enthält irgendeine Bitstelle des Kanalstatusbytes mit Ausnahme der Bitstelle 0 (Programmgesteuerte Unterbrechung) eine 1, so wird die Operation beendet. In gleicher Weise wird die Operation beendet, wenn irgendeine Bitstelle des Gerätestatusbytes mit Ausnahme von Kanalende und Geräteende (die eine 1 enthalten müssen) sowie der Statusänderung (die jedoch tatsächlich nur bei gewissen Operationen bei Magnetplattenspeichergeräten bzw. anderen Geräten mit Direktzugriff eine 1 enthalten kann) eine 1 enthält. Wenn eine Operation auf diese Weise abgebrochen wird, so wird auf den Kanal eine hängende Unterbrechung geschickt und die Adresse des nächsten CCW im CSW ist um 8 größer als die Adresse des zuletzt benutzten CCW (jedoch um 16 größer, wenn die Bitstelle „Statusänderung" eine 1 enthält). Wird die EA-Operation normal beendet, so kommt es zur Befehlskettung, wenn eine solche codiert ist.

Eine falsche Länge wird normalerweise dann angezeigt, wenn das Zählerfeld auf 0 abgebaut ist und Datenkettung nicht angegeben ist. Enthält die Bitstelle 34 des CCW eine 1 und die Bitstelle 32 eine 0, so unterbleibt die Anzeige einer falschen Satzlänge. Eine falsche Satzlänge wird auch dann angezeigt, wenn das Ende eines physikalischen Satzes erreicht ist und das Zählerfeld noch nicht auf 0 abgebaut ist. In diesem Fall wird die Anzeige ebenfalls dann unterdrückt, wenn die Bitstelle 32 auf 0 und die Bitstelle 34 auf 1 steht. Ist die Bitstelle 32 mit einer 1 versehen, so wird der Anzeiger für falsche Satzlänge nicht beachtet, so daß die Bedingung „Falsche Satzlänge" auftreten kann.

Der Anzeiger zur Unterdrückung von Datenübertragungen (Bitstelle 35) ist nur dann wirksam, wenn die Bitstelle niedrigster Ordnung des Kanalbefehlsschlüssels eine 0 enthält, d.h. bei den Befehlen Lesen, Lesen rückwärts und Prüfen. Enthält diese Bitstelle eine 1, so wird der Kanal veranlaßt, die Speicherung der Bytes, die er von der Einheit empfängt, im Hauptspeicher nicht vorzunehmen. Diese Bytes gehen einfach verloren. Sie werden jedoch genau so wie die anderen Bytes auf Gültigkeit geprüft. Durch sie kann somit die Bitstelle für Kanaldatenfehler auch auf 1 gesetzt werden.

Steht der Anzeiger für Programmunterbrechungen (Bitstelle 36) auf 1, so wird dadurch veranlaßt, daß die Bitstelle höchster Ordnung des Kanalstatusbytes auf 1 gesetzt wird. Damit kommt es zu einer Unterbrechung, die normalerweise dann stattfindet, sobald das CCW durch den Kanal entschlüsselt ist. Verursacht das CCW einen Programmfehler, so bleibt die Bitstelle 36 unbeachtet; die Unterbrechung erfolgt dann auf Grund der Beendigung des Kanalprogrammes.

---

1) Der Aufbau des Kanalstatusbytes wird im Abschnitt 5.9. beschrieben.

Unsere Beschreibung wäre nicht vollständig, wenn wir nicht auch die Bedeutung der Bitstelle 37 (indirekte Adressierung) behandeln würden. Enthält diese Bitstelle eine 1, so muß die im Kanalbefehlswort (CCW) stehende Adresse durch 4 teilbar sein. Die Daten werden nach Hauptspeicherplätzen gelesen, die bei der im indirekten Datenadreßwort (IDAW) angegebenen Adresse beginnen und bis zu einer Adresse reichen, die als nächste durch 2048 (bzw. 000 800 sedezimal) teilbar gefunden wird. Die Adresse des indirekten Datenadreßwortes ist die im CCW spezifizierte Adresse. Nach dieser ersten Phase wird das Einlesen fortgesetzt, wobei als nächstes indirektes Datenadreßwort dasjenige angesehen wird, das auf das in der CCW-Adresse angesprochene unmittelbar folgt. Die darin enthaltene Adresse muß durch 2048 teilbar sein. So viele indirekte Datenadreßwörter werden geholt, wie notwendig für die Durchführung der Eingabeoperation sind. Alle in ihnen enthaltenen Adressen mit Ausnahme der im ersten stehenden müssen durch 2048 teilbar sein. Bei Befehlen, die das Rückwärtslesen betreffen, müssen alle indirekten Adreßwörter mit Ausnahme des ersten eine Adresse beinhalten, die um 1 kleiner ist als eine durch 2048 teilbare Adresse. Hierbei werden die Datenzeichen in Hauptspeicherplätzen mit fallenden Adressen untergebracht, beginnend beim Platz, der durch die im ersten IDAW enthaltene Adresse gekennzeichnet ist. Die Einspeicherung setzt sich bis zu dem Speicherplatz fort, dessen Adresse durch 2048 teilbar ist; in diesem Moment wird das nächste IDAW herangezogen usw. Um dieses Vorgehen zu illustrieren, wollen wir ein gültiges Beispiel verfolgen:

	Inhalt	Bemerkungen
CCW	02 023008 04 000050	1. Kanalbefehlswort
Hauptspeicherplätze 023 008 bis 023 00B	000277F0	1. IDAW
Hauptspeicherplätze 023 00C bis 023 00F	00029000	2. IDAW
CCW	0C 023010 04 000050	2. Kanalbefehlswort
Hauptspeicherplätze 023 010 bis 023 013	0002903F	1. IDAW
Hauptspeicherplätze 023 014 bis 023 017	000277FF	2. IDAW

Abb. 5.9. Beispiele für Kanalbefehlswörter mit indirekter Adressierung (alle Inhalte sedezimal)

Das erste Kanalbefehlswort veranlaßt das Lesen von 16 Bytes in die Hauptspeicherplätze mit den Adressen 027 7F0 bis 027 7FF und von 64 Bytes in die Hauptspeicherplätze mit den Adressen 029 000 bis 029 03F. Das zweite Kanalbefehlswort bewirkt ein Rückwärtslesen zunächst von 64 Bytes in die Hauptspeicherplätze mit den Adressen 029 03F bis 029 000 und anschließend von 16 Bytes in die Hauptspeicherplätze von 027 7FF bis 027 7F0. Beide Kanalbefehlswörter füllen also die gleichen Hauptspeicherplätze mit Daten, das erste CCW durch Vorwärtslesen, das zweite durch Rückwärtslesen. Bei Schreiboperationen liegen die gleichen Zusammenhänge zwischen indirekten Datenadreßwörtern und Kanalbefehlen wie bei den eben beschriebenen Leseoperationen vor, nur daß es selbstverständlich kein Rückwärtsschreiben gibt.

Im Zählerfeld des CCW darf ein beliebiger von 0 verschiedener Wert angegeben werden. Enthält das Zählerfeld eine 0, so wird dadurch ein Programmfehler hervorgerufen, sogar bei Steuerungsbefehlen, die keine Datenübertragung nach sich ziehen. Eine Ausnahme hiervon bildet lediglich der Befehl „Verzweigen im Kanal" (TIC). Nur die ersten vier Bytes eines TIC-Befehles werden nämlich geprüft, die restlichen vier können beliebige Informationen aufweisen.

5.9. Das Kanalstatusbyte

Jedem Unterkanal ist ein Byte zugeordnet, das als Kanalstatusbyte bezeichnet wird. Dieses Byte enthält möglichst viele Informationen über den Zustand des Unterkanals. Der Programmierer kann dieses Statusbyte für Prüfzwecke verwenden. Die acht Bitstellen dieses Byte besitzen folgende Bedeutung (wir wollen sie mit S_0 und S_7 bezeichnen):

1) S_0 — Programmgesteuerte Unterbrechung
Wird durch jedes CCW auf 1 gesetzt, wenn dessen Bitstelle 36 eine 1 enthält. S_0 wird jedoch dann nicht auf 1 gesetzt, wenn das gleiche CCW die Bitstelle S_2 auf 1 setzt. Enthält S_0 eine 1, so tritt im Unterkanal eine hängende Unterbrechung auf.

2) S_1 — Falsche Länge
Wird auf 1 gesetzt, wenn das Ende eines physikalischen Satzes festgestellt wurde und entweder Datenkettung angegeben ist oder der Zähler ungleich Null ist und der Anzeiger für falsche Satzlänge im CCW auf 0 steht. S_1 wird ebenfalls auf 1 gesetzt, wenn das Ende eines physikalischen Satzes noch nicht erreicht wurde, obwohl der Zähler bereits auf 0 abgebaut ist und der Anzeiger für falsche Satzlänge nicht auf 1 steht. Bei einer 1 auf S_1 wird die Datenkettung unterdrückt, die Operation beendet und eine hängende Unterbrechung auf den Unterkanal geschickt.

3) S_2 — Programmfehler

S_2 wird in folgenden Fällen auf 1 gesetzt:

a) durch jedes CAW, dessen Bitstellen 4 bis 7 nicht auf 0 stehen,

b) durch jedes CAW, das auf den Bitstellen 8 bis 31 keine durch 8 teilbare Adresse aufweist,

c) durch jedes CAW, das auf den Bitstellen 8 bis 31 eine Adresse enthält, die für den Hauptspeicher des betreffenden Computers nicht existiert,

d) wenn das erste CCW eines Kanalprogrammes ein Befehl „Verzweigen im Kanal" ist,

e) wenn in einem Kanalprogramm zwei aufeinanderfolgende Befehle „Verzweigung im Kanal" vorliegen,

f) wenn ein CCW den Befehl „Verzweigen im Kanal" enthält und das Adreßfeld nicht durch 8 teilbar ist,

g) wenn ein CCW den Befehl „Verzweigen im Kanal" enthält und seine Adresse auf einen nicht existierenden Hauptspeicherplatz verweist,

h) wenn ein CCW einen Befehl enthält, der nicht gleich einem Befehl „Verzweigen im Kanal" ist und das Zählerfeld auf 0 gesetzt ist,

i) wenn ein CCW einen anderen Befehl als den Befehl „Verzweigen im Kanal" enthält und die Bitstellen 38 bis 39 nicht auf 0 stehen,

j) wenn ein CCW einen anderen Befehl als den Befehl „Verzweigen im Kanal" enthält und dieser entweder durch Befehlskettung oder als erster Befehl eines Kanalprogrammes angegeben wurde, sofern im CCW die Bitstellen 4 bis 7 Nullen aufweisen,

k) wenn ein CCW vorliegt, dessen Bitstelle 37 (indirekte Adressierung) auf 1 steht und dessen Adreßfeld (Bitstellen 8 bis 31) eine nicht durch 4 teilbare Adresse beinhaltet,

l) wenn eine Folge von indirekten Datenadreßwörtern auf eine nichtexistierende Hauptspeicheradresse aufläuft,

m) wenn ein indirektes Datenadreßwort (IDAW) auf eine nichtexistierende Hauptspeicheradresse verweist,

n) wenn die Bitstellen 0 bis 7 eines IDAW nicht gleich Null sind,

o) wenn ein IDAW mit Ausnahme des ersten nicht auf den Beginn oder beim Rückwärtslesen auf das Ende eines 2048 Stellen (Bytes) umfassenden Hauptspeicherblockes verweist,

p) wenn eine Datenadresse auftritt, die nicht existiert.

Zu nichtexistierenden Hauptspeicheradressen kommt es normalerweise dann, wenn eine Adresse gebildet wird, die die Hauptspeicherkapazität des Computers überschreitet. Jedoch brauchen bei einigen Modellen der Computerfamilie IBM /370 die Hauptspeicheradressen nicht in lückenloser Folge vorzuliegen. Das kann z.B. geschehen, wenn der Hauptspeicher aus einzelnen Teilen zusammengestellt wird und ein solches Teil vorübergehend nicht verfügbar ist.

Enthält S_2 eine 1, so wird die Operation unterdrückt und in den Unterkanal eine hängende Unterbrechung geschickt. Wenn diese Bedingung erst nach dem Starten einer EA-Operation festgestellt wird, so wird die Operation nach der Übertragung des nächsten Datenzeichens abgeschlossen.

4) S_3 — Speicherschutzfehler
S_3 wird auf 1 gesetzt, wenn der Versuch unternommen wird, bei einem Lese- oder Prüfbefehl bzw. bei einem Befehl „Lesen rückwärts" die Daten bzw. das Prüfbyte (die Prüfbytes) auf Speicherplätze zu speichern, deren Speicherschutzschlüssel ungleich dem im CAW angegebenen Schlüssel ist. Die Informationen werden nicht gespeichert. S_3 bleibt jedoch ohne Rücksicht auf den Speicherschutzschlüssel auf 0 stehen und die Informationen werden übertragen, wenn der Schlüssel im CAW gleich 0000 ist. Eine 1 auf S_3 unterdrückt das Speichern der Informationen, beendet die Operation und schickt eine hängende Unterbrechung auf den Unterkanal.

5) S_4 — Kanaldatenfehler
Wird auf 1 gesetzt, wenn ein Datenzeichen zwischen Kanal und Gerät übertragen wird, das eine gerade Anzahl von Einerbits aufweist. Enthält S_4 eine 1, so kann die Informationsübertragung fortgesetzt werden oder nicht fortgesetzt werden, aber eine Befehlskettung wird immer unterbunden. Bei einer Beendigung wird eine hängende Unterbrechung auf den Unterkanal geschickt. In einigen Fällen kann eine Zurücksetzung der Störung eintreten. Allen übertragenen Zeichen wird jedoch eine ungerade Anzahl von Einerbits aufgezwungen.

6) S_5 — Kanalsteuerfehler
S_5 wird auf 1 gesetzt, wenn die elektronischen Schaltkreise des Kanals fehlerhaft arbeiten. Der Status des Kanals kann nicht vorausgesagt werden; der Inhalt des CSW kann fehlerhaft sein. Der Kanal leitet eine Fehlerkorrekturfunktion ein.

7) S_6 — Interface-Steuerfehler [1]
S_6 wird in folgenden Fällen auf 1 gesetzt:

a) Ein EA-Gerät antwortet auf ein Kanalsignal in ungeeigneter Form.

b) Ein EA-Gerät wartet mit einer geeigneten Antwort auf ein Kanalsignal auf, doch weist diese Antwort Zeichen mit einer geraden Anzahl von Einerbits auf.

c) Ein EA-Gerät hat die Anwort über das maximal zulässige Zeitintervall hinaus verzögert.

d) Die Speichereinheit eines Multiplexkanals weist Fehler auf. (Unter der Speichereinheit des Multiplexkanals versteht man eine Speichereinheit, die das CSW, CCW usw. für jeden Unterkanal speichert.)

8) S_7 — Kettungsfehler
S_7 wird immer dann während einer Lese- oder Prüfoperation bzw. einer Operation „Lesen rückwärts" auf 1 gesetzt, wenn Datenkettung stattfindet und nach erfolgter Kettung das erste Byte bereits zu einem Zeitpunkt von der Einheit zur Verfügung gestellt wird, zu dem der Kanal es noch gar nicht behandeln kann. Zu dieser Überlaufbedingung kommt es deshalb, weil die Einheit die Daten schneller übertragen hat als der Kanal in der Lage war, sie zu empfangen. S_7 auf 1 gesetzt, beendet die Operation ohne weiteren Datentransport und schickt eine hängende Unterbrechung auf den Unterkanal.

Eine durch den Kanal selbst eingeleitete Fehlerkorrektur wird immer dann versucht, wenn die Bitstellen S_4, S_5 oder S_6 eine 1 enthalten. Hierdurch werden die Statusanzeiger im Kanal gelöscht, evtl. auch die eines oder mehrerer EA-Geräte, die an den Kanal angeschlossen sind. Wenn der Kanal eine Anzeigeleiste [2] besitzt, um dem Wartungspersonal zu helfen, so enthält die Bitstelle S_5 ebenfalls eine 1. Im allgemeinen sollte immer dann das Wartungspersonal bemüht werden, um die Anlage wieder instandzusetzen, wenn eine der Bitstellen S_5, S_6 oder S_7 auf 1 gesetzt wurde.

5.10. Starten Eingabe/Ausgabe (SIO)

Mit diesem Abschnitt wollen wir die Besprechung der EA-Instruktionen beginnen. Zuerst wollen wir die Instruktion „Starten Eingabe/Ausgabe" behandeln, zugleich als einzige in diesem Abschnitt.

Die Instruktion SIO liegt im S-Format vor:

a)

Starten Eingabe/Ausgabe — Start Input/Output	9C ¦ SI0

Auf der Bitstelle 15 muß bei dieser Instruktion eine 0 stehen. Die Bitstellen 8 bis 14 dieser Instruktion werden ignoriert und können infolgedessen beliebige Informationen enthalten. Bevor wir jedoch weitergehen, sollten wir zunächst einige neue Begriffe kennenlernen.

---

1) Unter dem Interface versteht man die Schnittstelle zwischen einem Kanal und einer Steuereinheit, d.h. die Stelle, an der die Steuereinheit mit dem Kanal verbunden ist.

2) Man beachte hierzu die Ausführungen im Abschnitt 5.4.

Unter einer Sofortoperation versteht man eine Kanaloperation, die unmittelbar nach der Entschlüsselung des Befehlsschlüssels abgeschlossen ist, so daß der Kanal sofort wieder frei für die Folgeoperation ist. Dabei werden keine Daten übertragen. Typische Sofortoperationen sind z.B.:

a) Auswahl eines Ablagefaches bei Kartenstanzern,
b) Starten des Zurückspulens eines Magnetbandes,
c) NOP[1].

Im Gerätestatusbyte wird hierbei 08, 0C oder 4C, sedezimal ausgedrückt, zurückgegeben, wenn die Operation zufriedenstellend abgelaufen ist. Das heißt also, daß Kanalende, möglicherweise auch Geräteende und bei Anzeige von Geräteende zusätzlich evtl. auch noch eine Statusänderung signalisiert wird.

Bei einigen EA-Geräten ist eine Einrichtung vorhanden, die man als Befehlswiederholung zu bezeichnen pflegt. Wenn das EA-Gerät unfähig ist, die Operation zu vervollständigen, so kann der Befehl vom Kanal wiederholt werden, ohne daß die Steuerung an die ZE zurückgegeben wird. Das EA-Gerät gibt hierbei ein Gerätestatusbyte [2] zurück, das die sedezimalen Werte 4A oder 4E besitzt, d.h. es werden folgende Bitstellen auf jeden Fall auf 1 gesetzt: Statusänderung, Kanalende und Gerätefehler. Hinzu kommt u.U. die Bitstelle U_5 für die Anzeige des Geräteendes. Ist das Geräteende angezeigt, so wiederholt der Kanal sofort den Befehl; im anderen Fall wartet der Kanal eine von dem betreffenden Gerät verursachte Unterbrechung ab, durch die das Geräteende mitgeteilt wird. Bei Wiederholung eines Kanalbefehlswortes (CCW) wird dieses erneut vom Hauptspeicher geholt, so daß jederzeit eine erneute Programmunterbrechung (Bitstelle 36 des CCW) eintreten kann. Bei der erstmaligen Ausführung von Eingabeoperationen können außerdem Daten inkorrekt gespeichert werden. Die als Resultat einer letzten Ausführung der Eingabeoperation gespeicherten Daten sind jedoch korrekt, es sei denn, daß die Befehlswiederholung zu einem Fehler führte. In diesem Fall entsteht eine reguläre Unterbrechung der ZE.

Wie alle Instruktionen der IBM /370 wird auch die Instruktion SIO schrittweise ausgeführt. Nach jedem einzelnen Schritt erfolgt eine Prüfung. Tritt eine entsprechende Bedingung auf, so wird der Bedingungsschlüssel gesetzt und die Operation beendet. Tritt die Bedingung dagegen nicht auf, so wird die Ausführung mit dem nächsten Schritt fortgesetzt. Wird die Instruktion SIO ausgeführt, so wird stets der Bedingungsschlüssel gesetzt. Wenn bei der Ausführung einer SIO-Instruktion im Kanal eine gefüllte Anzeigeleiste vorhanden ist, so wird das Kanalstatuswort (CSW) mit Ausnahme der Bitstelle 5 (A) und der Bitstelle S_5 gelöscht; diese beiden Bitstellen bleiben auf 1 gesetzt. Die Anzeigeleiste selbst bleibt bestehen und die SIO-Instruktion wird beendet, wobei der Bedingungsschlüssel den Wert 1 zugewiesen erhält. In allen anderen Fällen bleibt das CSW ungeändert, es sei denn, daß der Bedingungsschlüssel 1 auftritt. Dann werden nämlich die beiden Statusbytes ersetzt, der Rest des CSW bleibt jedoch unberührt.

Logisch gesehen läuft die Instruktion SIO in folgenden Einzelschritten ab:

1) Ist der Kanal nicht arbeitsfähig (betriebsbereit), so wird der BS[3] auf 3 gesetzt.

2) Arbeitet der Kanal im Einpunktbetrieb, so wird der BS auf 2 gesetzt.

---

1) NOP ist die Abkürzung von no operation (keine Operation).
2) Siehe hierzu Abschnitt 5.6.
3) BS soll in den Abschnitten 5.10. bis 5.15. als Abkürzung von Bedingungsschlüssel verstanden werden.

3) (Gilt nur für die Multiplexkanäle.) Wenn der Unterkanal nicht arbeitsfähig (betriebsbereit) ist, so wird der BS auf 3 gesetzt.

4) Hängt in einem Unterkanal eine Unterbrechung, so wird der BS auf 2 gesetzt.

5) (Gilt nur für die Multiplexkanäle.) Arbeitet der Kanal im Mehrpunktbetrieb, so wird der BS auf 2 gesetzt.

6) Das Kanaladreßwort (CAW) wird wie folgt überprüft:
   a) Enthalten die Bitstellen 4 bis 7 irgendeine 1?
   b) Enthalten die Bitstellen 29 bis 31 irgendeine 1?
   Muß irgendeine dieser Fragen bejaht werden, so wird die Bitstelle S_2 des Kanalstatusbytes auf 1 gesetzt. Dem BS wird der Wert 1 zugewiesen und die beiden Statusbytes werden ins CSW gespeichert, wobei das Gerätestatusbyte gleich 0 ist.

7) Wenn ein Kanalsteuerfehler festgestellt wird, so wird die Bitstelle S_2 des Kanalstatusbytes auf 1 gesetzt. BS und CSW werden wie im Schritt 6) behandelt.

8) Wenn eine nichtexistente Hauptspeicheradresse im CAW vorliegt, so wird die Bitstelle S_2 des Kanalstatusbytes auf 1 gesetzt. BS und CSW werden wie im Schritt 6) behandelt.

9) Das CCW wird in folgender Hinsicht überprüft:
   a) Enthalten die Bitstellen 5 bis 7 irgendeine 1?
   b) Enthalten die Bitstellen 38 bis 39 irgendeine 1?
   c) Enthalten die Bitstellen 48 bis 63 (Zählerfeld) lauter Nullen?
   d) Enthält Bitstelle 37 eine 1 und sind die Bitstellen 30 und 31 (letzte beiden Bitstellen der Adresse) nicht auf 0 gesetzt?
   e) Enthält Bitstelle 37 eine 1 und bezieht sich das erste indirekte Adreßwort auf eine nichtexistente Adresse?
   f) Enthält Bitstelle 37 eine 1 und enthalten die Bitstellen 0 bis 7 des ersten indirekten Adreßwortes irgendeine 1?
   Muß irgendeine dieser Fragen bejaht werden, dann wird S_2 auf 1 gesetzt. BS und CSW werden wie im Fall 6) behandelt.

10) Enthält die Bitstelle 36 eine 1, dann wird die Bitstelle S_0 des Kanalstatusbytes auf 1 gesetzt. Dadurch wird eine Unterbrechung hervorgerufen, die im Unterkanal hängenbleibt; die Ausführung des Kanalprogrammes wird jedoch anderweitig nicht tangiert.

11) Ist ein Interface-Steuerfehler vorhanden, so wird S_6 auf 1 gesetzt, der BS ebenfalls auf 1. Die beiden Statusbytes werden ins CSW gespeichert; dabei ist das Gerätestatusbyte gleich Null.

12) Ist das EA-Gerät nicht arbeitsfähig (betriebsbereit), so wird der BS auf 3 gesetzt.

13) Hängt in der Steuereinheit eine Unterbrechung für ein anderes Gerät, so werden die Bitstellen U_1 und U_3 des Gerätestatusbytes auf 1 gesetzt, alle anderen auf 0, d.h. das Gerätestatusbyte nimmt den sedezimalen Wert 50 an. Dem BS wird außerdem der Wert 1 zugewiesen; Gerätestatusbyte und Kanalstatusbyte werden ins CSW gebracht.

14) Wenn die Steuereinheit beschäftigt ist, so erfolgt die gleiche Reaktion wie bei Schritt 13).

15) Wenn das EA-Gerät beschäftigt ist, so wird dem Gerätestatusbyte der Wert 10 (sedezimal) zugewiesen, d.h. der Bitstelle U_3 eine 1. BS und CSW werden wie bei Schritt 13) behandelt.

16) Ist ein Interface-Steuerfehler aufgetreten, so wird die Bitstelle S_6 des Kanalstatusbytes auf 1 gesetzt. BS und CSW werden wie im Fall 6) behandelt.

17) Der Bedingungsschlüssel BS wird auf 0 gesetzt, wenn eine Befehlswiederholung vorliegt, d.h. das Gerätestatusbyte nimmt den sedezimalen Wert 4A oder 4E an.

18) Wenn das Gerätestatusbyte den Wert 00 aufweist, wird der Bedingungsschlüssel auf 0 gesetzt.

19) Wenn das Gerätestatusbyte einen der sedezimalen Werte 08, 0C oder 4C angenommen hat und ferner auf den Bitstellen 32 und 33 (Kettungsbitstellen) jeweils eine 0 steht, so wird der BS auf 1 gesetzt und beide Statusbytes ins CSW gespeichert.

20) In allen anderen Fällen wird der BS auf 1 gesetzt sowie das CSW mit den Werten der beiden Statusbytes versorgt.

Beim Auftreten der Bedingungsschlüssel 2 oder 3 wird die EA-Operation abgeschlossen, ohne daß sie gestartet wurde. Bei einem Wert des BS von 0 wird die Operation begonnen. Ist der Bedingungsschlüssel gleich 1 und enthält die Bitstelle U_4 (Kanalende) des Gerätestatusbytes eine 1, so wurde eine Sofortoperation abgeschlossen. In allen anderen Fällen, in denen der BS gleich 1 ist, wird die Operation nicht begonnen.

5.11. Starten Eingabe/Ausgabe mit schneller Freigabe (SIOF)

Die Instruktion

Starten Eingabe/Ausgabe mit schneller Freigabe — Start Input/Output fast Release	9C	SIOF

unterscheidet sich formal von der SIO-Instruktion dadurch, daß die Bitstelle 15 eine 1 enthält. Sie wird fast in der gleichen Weise wie die SIO-Instruktion ausgeführt. Bei Blockmultiplexkanälen, bei denen die Bitstelle 0 des PSW oder das Steuerregister 0 eine 1 enthält, wird diese Instruktion in den ersten sieben Schritten logisch genau so ausgeführt wie die Instruktion SIO. An Stelle der Schritte 8) bis 20) wird jedoch nur der folgende Schritt ausgeführt:

8) In allen anderen Fällen wird der BS auf 1 gesetzt.

Dieser Ablauf führt mitunter dazu, daß der BS gleich 0 ist, während er bei einer SIO-Instruktion den Wert 1 oder 3 angenommen hätte. Mit Ausnahme des Falles, der durch den Schritt 19) der SIO-Instruktion beschrieben wurde, erzeugt der Kanal eine Unterbrechung, wobei der BS auf 1 oder 1 gesetzt wird. Die Statusbytes werden genau so behandelt, als ob eine SIO-Instruktion stattgefunden hätte. (Abgesehen davon besteht die Möglichkeit des Auftretens eines zusätzlichen Kanalsteuerfehlers oder eines Interface-Steuerfehlers, weil es in der Maschinenausrüstung erst dann zu Fehlern zu kommen braucht, wenn diese Instruktion bereits gestartet wurde.) Bei einem Fall, der der Bedingung des Schrittes 19 des Ablaufes der SIO-Instruktion (siehe Abschnitt 5.10.) entspricht, führt die SIOF-Instruktion jedoch nicht zu einer Unterbrechung mit dem BS gleich 1, sondern es ereignet sich eine reguläre Unterbrechung mit dem BS gleich 0.

Bei einigen Modellen der Computerfamilie IBM /370 wird die SIOF-Instruktion genau so ausgeführt wie die SIO-Instruktion. Bei Bytemultiplexkanälen und bei Selektorkanälen läuft die SIOF-Instruktion genau so ab wie die SIO-Instruktion. Der Vorteil der SIOF-Instruktion ist in ihrer Ablaufgeschwindigkeit zu sehen. Die Ausführung einer SIO-Instruktion sperrt die ZE solange gegenüber anderen Operationen, bis der

Kanal ein Statusbyte vom EA-Gerät geholt hat, d.h. bis etwa 25 μs nach dem Entschlüsseln und Prüfen des Kanaladreßwortes (CAW). Die SIOF-Instruktion hingegen spart diese Zeit bei der ZE ein, wobei freilich damit spekuliert werden muß, daß eine zusätzliche EA-Unterbrechung auftreten kann, wenn der Bedingungsschlüssel die Werte 1 oder 3 annimmt. Da die SIOF-Instruktion bei den schnelleren Modellen der Computerfamilie IBM /370 ungefähr 2 μs dauert, bei den langsameren ungefähr 30 μs, so erlangt die Einsparung von 25 μs bei der Kanalzeit, die für eine SIO-Instruktion erforderlich ist, doch eine erhebliche Bedeutung bei den schnelleren Modellen. Die langsameren Modelle führen die SIOF-Instruktion wie die SIO-Instruktion aus.

5.12. Halten Eingabe/Ausgabe

Zwei Instruktionen sind verfügbar, um die Ausführung eines Kanalprogrammes abzuschließen. Sie lauten:

a)

Halt Eingabe/Ausgabe — Halt Input/Output	9E	HIO

b)

Halt Gerät — Halt Device	9E	HDV

Beide Instruktionen besitzen das S-Format. Formal unterscheiden sie sich in der Bitstelle 15: HIO weist dort eine 0 auf, HDV dagegen eine 1.

Wenn eine gefüllte Anzeigeleiste im Kanal vorliegt, dann wird bei beiden Instruktionen das Kanalstatuswort (CSW) mit Ausnahme der Bitstellen 5 (A) und S_5 auf Null gesetzt, siehe hierzu auch Abb. 5.6. im Abschnitt 5.4.; diese beiden Bitstellen enthalten jeweils eine 1. Die Anzeigeleiste selbst bleibt bestehen und der Bedingungsschlüssel wird auf 1 gesetzt. In allen anderen Fällen bleibt das CSW ungeändert, es sei denn, daß der Bedingungsschlüssel 1 auftritt. Dann nämlich werden die beiden Statusbytes des CSW ersetzt.

Die Instruktion HIO läuft in folgenden Einzelschritten ab:

1) Ist der Kanal nicht betriebsbereit, so wird der BS auf 3 gesetzt.

2) Arbeitet der Kanal im Einpunktbetrieb, so wird der BS auf 2 gesetzt.

3) (Gilt nur für die Multiplexkanäle.) Wenn der angesprochene Unterkanal nicht arbeitsfähig (betriebsbereit) ist, wird der BS auf 3 gesetzt.

4) Hängt im angesprochenen Unterkanal eine Unterbrechung, so wird der BS auf 0 gesetzt.

5) Bei Feststellung eines Kanalsteuerfehlers wird die Bitstelle S_5 des Kanalstatusbytes auf 1 gesetzt. Ferner wird dem BS der Wert 1 zugewiesen und ins CSW beide Statusbytes gespeichert, wobei das Gerätestatusbyte gleich Null ist.

6) Bei Feststellung eines Interface-Steuerfehlers wird die Bitstelle S_6 des Kanalstatusbytes auf 1 gesetzt. BS und CSW werden wie bei Schritt 5) behandelt.

7) Wenn das angesprochene EA-Gerät nicht arbeitsfähig (betriebsbereit) ist, wird der BS auf 3 gesetzt.

8) Wenn die Steuereinheit beschäftigt ist, so wird dem Gerätestatusbyte der sedezimale Wert 50 zugewiesen, d.h. die Bitstellen U_1 und U_3 bekommen eine 1. Der Bedingungsschlüssel erhält den Wert 1; ferner werden beide Statusbytes ins CSW weggespeichert.

9) In allen anderen Fällen wird der BS auf 1 gesetzt und die Plätze für die beiden Statusbytes im CSW werden gelöscht, d.h. auf Null gebracht.

Der Bedingungsschlüssel wird in den gleichen Situationen mit dem Wert 3 versorgt wie bei der Instruktion SIO. Er wird auf 2 gesetzt, wenn eine Operation im Einpunktbetrieb im Gang ist; in einem solchen Fall wird die Datenübertragung sofort angehalten, doch das Gerät läuft bis zu einer normalen Stoppstelle weiter. Der BS wird nur dann auf 0 gesetzt, wenn im Unterkanal eine Unterbrechung ansteht; in diesem Fall läuft gerade keine Operation ab, so daß auch keine angehalten zu werden braucht.

Wird der BS auf 1 oder 3 gesetzt, so wird dem Kanal aufgetragen, daß er den Haltebefehl in dem Moment hinausschickt, in dem das Gerät betriebsbereit wird oder wenn beim Bytemultiplexkanal das Gerät die Übertragung des nächsten Datenbytes anfordert.

Bei einem Selektorkanal erzeugt die HIO-Instruktion stets aus eigenem Antrieb eine hängende Unterbrechung. Bei den anderen Kanalarten kommt es zu keiner eigenständigen Unterbrechung, hier müssen Kanalende, Geräteende oder Steuereinheitenende auftreten, um eine Unterbrechung zu bewirken. Wenn jedoch ein gemeinsam benutzter Unterkanal eines Multiplexkanals adressiert wird, so wird durch HIO jede in Gang befindliche Datenübertragungsoperation auf dem Unterkanal abgebrochen, sogar wenn diese zu einem anderen Gerät als dem adressierten erfolgt.

Die Instruktion HDV unterscheidet sich von der Instruktion HIO in einigen Punkten, wenn sie unter den nachfolgenden Gegebenheiten hinausgeschickt wird:

a) bei einem Selektorkanal zu einem Gerät, mit dem der Kanal zur fraglichen Zeit nicht arbeitet,
b) bei einem Bytemultiplexkanal, wenn ein Einpunktbetrieb im Gange ist,
c) bei einem Bytemultiplexkanal zu einem Gerät, dessen zugehöriger Unterkanal zur fraglichen Zeit gerade ein anderes Gerät versorgt.

In diesen drei Fällen stoppt die Instruktion HDV die ablaufende Operation nicht ab.

Die Instruktion HDV zerfällt logisch gesehen in folgende einzelnen Schritte:

1) Ist der Kanal nicht arbeitsfähig (betriebsbereit), so wird der BS auf 3 gesetzt.

2) (Nur bei einigen Modellen der Computerfamilie IBM /370.) Wenn der Kanal im Einpunktbetrieb mit dem adressierten Gerät arbeitet, so wird der BS auf 2 gesetzt.

3) (Nur bei einigen Modellen.) Wenn der Kanal im Einpunktbetrieb mit einem anderen als dem adressierten Gerät arbeitet und der Unterkanal nicht arbeitsfähig (betriebsbereit) ist, dann wird der BS auf 2 gesetzt.

4) (Nur bei einigen Modellen.) Wenn der Kanal im Einpunktbetrieb arbeitet und eine Unterbrechung im Unterkanal hängt, dann wird der BS auf 2 gesetzt.

5) (Nur bei einigen Modellen.) Wenn sowohl Kanal als auch Unterkanal im Einpunktbetrieb mit einem nicht adressierten Gerät arbeiten, wird der BS auf 2 gesetzt.

6) Wenn der Unterkanal nicht arbeitsfähig (betriebsbereit) ist, wird der BS auf 3 gesetzt.

7) Wenn eine Unterbrechung im Unterkanal hängt, wird der BS auf 0 gesetzt.

8) Bei Feststellung eines Kanalsteuerfehlers wird die Bitstelle S_5 des Kanalstatusbytes auf 1 gesetzt. Dem BS wird der Wert 1 zugewiesen und ins CSW werden beide Statusbytes gespeichert, wobei das Gerätestatusbyte gleich Null ist.

9) Bei Feststellung eines Interface-Steuerfehlers wird die Bitstelle S_6 des Kanalstatusbytes auf 1 gesetzt. BS und CSW werden wie bei Schritt 8) behandelt.

10) Wenn der Unterkanal mit einem anderen als dem adressierten Gerät arbeitet, wird der BS auf 0 gesetzt.

11) Wenn der Kanal im Einpunktbetrieb mit einem anderen als dem adressierten Gerät arbeitet, wird der BS auf 2 gesetzt.

12) Bei Feststellung eines Interface-Steuerfehlers wird wie bei Schritt 9) verfahren.

13) Wenn das EA-Gerät nicht arbeitsfähig (betriebsbereit) ist, wird dem BS der Wert 3 zugewiesen.

14) Wenn die Steuereinheit beschäftigt ist, wird das Gerätestatusbyte auf den sedezimalen Wert 50 gebracht, d.h. den Bitstellen U_1 und U_3 wird eine 1 zugewiesen. Der Bedingungsschlüssel erhält den Wert 1. Ferner werden beide Statusbytes ins CSW weggespeichert.

15) In allen übrigen Fällen wird der BS auf 1 gesetzt. Außerdem werden die Plätze für die beiden Statusbytes im CSW gelöscht, d.h. auf Null gebracht.

Einige Kanäle können die Instruktion HDV nicht ausführen. Bei ihnen läuft deshalb diese Instruktion wie eine HIO-Instruktion ab.

5.13. Prüfen Kanal (TCH)

Die Instruktion „Prüfen Kanal", für die das S-Format gilt, lautet genau wie folgt:

Prüfen Kanal — Test Channel	9F	TCH

Auf ihren Bitstellen 8 bis 15 können beliebige Informationen stehen, da der Inhalt dieser Bitstellen ignoriert wird.

Die TCH-Instruktion wird wie folgt ausgeführt:

1) Ist der Kanal nicht betriebsbereit, so wird der BS auf 3 gesetzt.

2) Arbeitet der Kanal im Einpunktbetrieb, so wird der BS auf 2 gesetzt.

3) Steht im Kanal eine Unterbrechung an oder ist die Anzeigeleiste im Kanal gefüllt, so wird der BS auf 1 gesetzt.

4) Wenn der Kanal verfügbar ist, wird der BS auf 0 gesetzt.

Bei der Instruktion TCH wird nur der Kanalteil der EA-Adresse entschlüsselt. Weder die EA-Geräte noch die Steuereinheiten noch die Unterkanäle werden in die ablaufende Operation verwickelt.

5.14. Prüfen Eingabe/Ausgabe (TIO)

Die Instruktion

Prüfen Eingabe/Ausgabe — Test Input/Output	90	TIC

im S-Format wird benutzt, um Unterbrechungen selektiv bearbeiten zu können. Die Bitstelle 15 dieser Instruktion weist eine 0 auf.

Wenn der Bedingungsschlüssel gleich 1 ist, so hat diese Instruktion das gesamte Kanalstatuswort (CSW) gespeichert. Die Ergebnisse sind die gleichen, die erzielt worden wären, wenn eine Unterbrechung zum Zuge gekommen wäre, die das CSW speichert. Der Wert 3 für den BS zeigt an, daß ein Kanal, ein Unterkanal oder ein Gerät nicht arbeitsfähig (betriebsbereit) war. Der Bedingungsschlüssel mit dem Wert 2 bedeutet schließlich, daß ein Kanal beschäftigt ist oder eine Unterbrechung hängt. Besitzt der Bedingungsschlüssel den Wert 0, so wird damit ein unbeschäftigtes Gerät kenntlich gemacht.

Wie bei den anderen EA-Instruktionen erfolgt bei einer gefüllten Anzeigeleiste im Kanal eine Speicherung von Informationen ins CSW. Die Bitstellen 5(A) und S_5 des Kanalstatusbytes erhalten hierbei den Wert 1, alle anderen werden auf Null gebracht. Dem Bedingungsschlüssel wird der Wert 1 zugewiesen. Jede weitere Verarbeitung wird eingestellt. In allen anderen Fällen läuft die TIO-Instruktion, logisch gesehen, mit folgenden Einzelschritten ab:

1) Wenn der Kanal nicht arbeitsfähig (betriebsbereit) ist, so wird der BS auf 3 gesetzt.

2) (Gilt nur für einige Arten von Multiplexkanälen.) Wenn der Kanal im Einpunktbetrieb arbeitet und der Unterkanal eine hängende Unterbrechung für das adressierte EA-Gerät aufweist, so wird der BS auf 1 gesetzt und das komplette CSW gespeichert; der Adreßteil übernimmt hierbei die Adresse aus dem Kanaladreßwort (CAW).

3) Wenn der Kanal im Einpunktbetrieb arbeitet, wird der BS auf 2 gesetzt.

4) (Gilt nur für Multiplexkanäle.) Wenn der Unterkanal nicht arbeitsfähig (betriebsbereit) ist, wird der BS auf 3 gesetzt.

5) Wenn eine hängende Unterbrechung im Unterkanal für das adressierte EA-Gerät festgestellt wird, so wird der BS auf 1 gesetzt und das komplette CSW gespeichert; d.h. wie bei Schritt 2).

6) Wenn eine hängende Unterbrechung im Unterkanal für ein nicht adressiertes EA-Gerät festgestellt wird, so wird der BS auf 2 gesetzt.

7) (Gilt nur für Multiplexkanäle.) Wenn der Unterkanal beschäftigt ist, wird der BS auf 2 gesetzt.

8) Wenn ein Kanalsteuerfehler entdeckt wird, so wird die Bitstelle S_5 der Kanalstatusbytes auf 1 gesetzt. Dem BS wird der Wert 1 zugewiesen. Außerdem wird das komplette CSW gespeichert (Zähler und Gerätestatusbyte sind dabei gelöscht, d.h. jede ihrer Bitstellen enthält Null). Der Adreßteil im CSW ist ebenfalls gelöscht.

9) Ist ein Interface-Steuerfehler vorhanden, dann wird der Bitstelle S_6 des Kanalstatusbytes der Wert 1 zugewiesen. BS und CSW werden wie bei Schritt 8) behandelt.

10) Ist das adressierte EA-Gerät nicht arbeitsfähig (betriebsbereit), so wird der BS auf 3 gesetzt.

11) Ist die Steuereinheit beschäftigt, so wird dem Gerätestatusbyte der sedezimale Wert 50 zugewiesen, d.h. die Bitstellen U_1 und U_2 werden auf 1 gesetzt. Dem BS wird der Wert 1 zugewiesen. Gerätestatusbyte und Kanalstatusbyte werden ins CSW gebracht. Der Zähler des CSW ist gelöscht, desgleichen der Adreßteil.

12) Ist ein Interface-Steuerfehler vorhanden, so werden die gleichen Maßnahmen durchgeführt wie bei Schritt 9).

13) Ist das EA-Gerät beschäftigt, dann wird das Gerätestatusbyte auf 10 (sedezimal) gesetzt, d.h. der Bitstelle U_3 eine 1 zugewiesen. Der BS wird auf 1 gebracht. Das CSW wird wie bei Schritt 11) behandelt.

14) Wenn eine Unterbrechung beim EA-Gerät hängt, d.h. nicht alle Bitstellen des Gerätestatusbytes sind mit Nullen versehen, so wird der BS auf 1 gesetzt. Das CSW wird wie bei Schritt 11) behandelt.

15) In allen anderen Fällen wird der BS auf 0 gesetzt.

Bei einigen EA-Geräten können die Unterbrechungen durch die Instruktion TIO nicht behoben werden. Solche Geräte antworten im Schritt 14) mit der Wertzuweisung von 40 (sedezimal) zum Gerätestatusbyte, d.h. der Bitstelle U_1 wird auf 1 gebracht. Immer, wenn die Adresse aus dem CAW ins CSW gespeichert wird, bedeutet das die Aufhebung der Unterbrechung.

5.15. Aufheben der Eingabe/Ausgabe (CLRIO)

Die Instruktion

Aufhebung der Eingabe/Ausgabe —— Clear Input/Output	9D	CLRIO

im S-Format beendet die laufende Operation des adressierten EA-Gerätes und veranlaßt die Speicherung des Operationszustandes ins CSW im Augenblick der Beendigung der Operation. Die Bitstelle 15 dieser Instruktion enthält eine 1.

Wie bei der Instruktion TIO (siehe Abschnitt 5.14.) wird das gesamte CSW mit Informationen versorgt. Liegt im Kanal eine gefüllte Anzeigeleiste vor, so wirkt die Instruktion CLRIO wie die Instruktion TIO. In allen anderen Fällen läuft die Ausführung von CLRIO in folgenden Einzelschritten ab:

1) Ist der Kanal nicht arbeitsfähig (betriebsbereit), so wird der BS auf 3 gesetzt.

2) (Gilt nur für einige Arten von Kanälen.) Wenn der Kanal im Einpunktbetrieb arbeitet, wird der BS auf 2 gesetzt.

3) (Gilt nur für Multiplexkanäle.) Wenn der Unterkanal nicht arbeitsfähig (betriebsbereit) ist, wird der BS auf 3 gesetzt.

4) Wenn eine hängende Unterbrechung im Unterkanal für das adressierte EA-Gerät festgestellt wird, so wird der BS auf 1 gesetzt. Ins CSW werden die Adresse des CAW, beide Statusbytes und der Zähler gespeichert, d.h. das komplette CSW wird mit Werten versorgt.

5) Wenn der Unterkanal mit dem adressierten EA-Gerät arbeitet, so wird die laufende Operation nicht mehr fortgesetzt, sondern beendet. BS und CSW werden wie im Schritt 4) behandelt.

6) Wird ein Kanalsteuerfehler entdeckt, so wird die Bitstelle S_5 des Kanalstatusbytes auf 1 gesetzt. Dem BS wird der Wert 1 zugewiesen. Außerdem wird das komplette CSW gespeichert, wobei Zähler, Gerätestatusbyte und Adreßteil gelöscht sind, d.h. alle ihre Bitstellen enthalten eine Null.

7) Wird ein Interface-Steuerfehler entdeckt, so wird die Bitstelle S_6 des Kanalstatusbytes auf 1 gesetzt. BS und CSW werden wie bei Schritt 6) behandelt.

8) In allen anderen Fällen wird der BS auf 0 gesetzt.

Einige Kanaltypen sind nicht fähig, die Instruktion CLRIO auszuführen. Sie behandeln sie deshalb wie die Instruktion TIO.

Abschließend wollen wir zusammenfassen, welche Werte der Bedingungsschlüssel bei der Ausführung der Instruktion CLRIO annehmen kann. Der BS wird auf 3 gesetzt, wenn entweder Kanal oder Unterkanal nicht arbeitsfähig (betriebsbereit) sind, auf 2 bei einem in Arbeit befindlichen Kanal, auf 0 bei einem unbeschäftigten EA-Gerät und auf 1 bei einem in Arbeit befindlichen EA-Gerät oder bei einem EA-Gerät mit hängender Unterbrechung.

5.16. Ein Beispiel für die Programmierung der EA

Um ein besseres Verständnis darüber zu erreichen, wie die Zusammenarbeit zwischen Kanälen und der ZE abläuft und wie die Instruktion TIO wirksam eingesetzt werden kann, wollen wir ein Programmsystem zusammenstellen, das ein Lochkartenpaket liest und stanzt. Das Programmsystem wird aus vier einzelnen Programmen gebildet werden, drei davon werden als Kanalprogramme verfaßt. Das vierte ist zum Ablauf in der ZE bestimmt, wir wollen es in den folgenden Betrachtungen konsequent als Problemprogramm bezeichnen, weil es zur Lösung der gestellten Aufgabe dient.

Die zu lösende Aufgabe ist wie folgt formuliert: Auf den Spalten 13 bis 16 der Eingabekarten ist ein x-Wert und auf den Spalten 75 bis 80 ein y-Wert abgelocht. Im Programm soll der Absolutbetrag der Summe von x und y berechnet werden und auf die Spalten 1 bis 8 der gleichen Karten gestanzt werden. Um bei einem Beispiel zu bleiben: Nehmen wir an, daß auf einer Karte der Wert 1234 auf den Spalten 13 bis 16 und der Wert bb9765 auf den Spalten 75 bis 80 abgelocht ist [1]. Dann soll im Programm die Summe aus 1234 und aus 9765 errechnet werden; auf die Spalten 1 bis 8 dieser Karte ist dann dieser Wert (00010999) zu stanzen.

Wir werden unsere einzelnen Programme in sedezimaler Form schreiben. Wir wollen für den Ablauf der Programme weiterhin voraussetzen, daß die Programme bereits geladen sind und daß die Register folgende Werte enthalten:

Register 1	0000 2100
Register 2	0000 2200
Register 3	0000 2300
Register 4	beliebig
Register 5	beliebig

Damit wir von unserem in 5.3. besprochenen Kanalprogramm Gebrauch machen können, werden wir den einzelnen Daten folgende Speicherplätze zuordnen:

ØØ2 1ØØ bis ØØ2 1Ø3	Eingelesener x-Wert
ØØ2 1Ø4 bis ØØ2 1Ø9	Eingelesener y-Wert
ØØ2 1ØA bis ØØ2 111	Auszugebende Summe (z-Wert)
ØØ2 112 bis ØØ2 114	Gepackter x-Wert (0x xx x±)
ØØ2 115 bis ØØ2 118	Gepackter y-Wert (0y yy yy y±)

Für die Kanalbefehlswörter (CCW), für die Kanaladreßwörter (CAW) und für benötigte Konstanten wollen wir die Speicherplätze ab 002 200 verwenden; die Instruktionen unseres Problemprogrammes sollen ab Speicherplatz 002 300 gespeichert werden. Wir

1) b bedeutet hier das Leerzeichen.

wollen ferner voraussetzen, daß zu Beginn dieses Programmes das laufende PSW den Inhalt 0000 0000 0000 2300 aufweist, so daß alle Unterbrechungen maskiert sind und die erste Instruktion vom Speicherplatz 002 300 geholt wird [1]. Zum Schluß wollen wir noch eine letzte Annahme erwähnen: Das Kartengerät IBM 2520 ist an den Bytemultiplexkanal angeschlossen und besitzt die EA-Adresse 00D.

Wir sollten uns noch daran erinnern, daß das wirksame CAW die Adresse 000 048 besitzen muß und daß die Adressen der Statusbytes im CSW 000 044 (Gerätestatus) und 000 045 (Kanalstatus) sind.

ØØ2 2ØØ	Ø9 ØØ21ØA 6Ø ØØØØØ8
ØØ2 2Ø8	C2 ØØØØØØ 9Ø ØØØØØC
ØØ2 21Ø	ØØ ØØ21ØØ 8Ø ØØØØØ4
ØØ2 218	ØØ ØØØØØØ 9Ø ØØØØ3A
ØØ2 22Ø	ØØ ØØ21Ø4 ØØ ØØØØØ6

Das erste Kanalprogramm muß praktisch die gleichen Funktionen erfüllen wie das zweite mit einer einzigen Ausnahme: Es darf keine Karte stanzen. Deshalb können wir zur Erfüllung der Aufgaben des ersten Kanalprogrammes einfach das zweite Kanalprogramm verwenden, wenn wir es ab Speicherplatz 002 208 ablaufen lassen. Beim dritten Programm ist nur das Stanzen einer Karte erforderlich; es ist also die Operation auszuführen, die von dem CCW auf 002 200 erledigt wird. Im Fall der letzten Karte müssen wir jedoch den Befehlsschlüssel 01 verwenden, da die Karte sowohl zugeführt als auch gestanzt werden muß. Es darf ferner auch keine Datenkettung angegeben werden. Unser drittes Kanalprogramm wird infolgedessen aus einem einzigen CCW bestehen, das wir auf dem nächsten verfügbaren Speicherplatz speichern wollen:

ØØ2 228	Ø1 ØØ21ØA 2Ø ØØØØØ8

Wir benötigen weiterhin drei Kanaladreßwörter (CAW). Wir wollen sie hintereinander auf die nächsten freien Speicherplätze speichern, von wo aus wir sie auf den Hauptspeicherplatz 000 048 dann übertragen, wenn das erforderlich ist.

ØØ2 23Ø	ØØ ØØ2 2Ø8
ØØ2 234	ØØ ØØ2 2ØØ
ØØ2 238	ØØ ØØ2 228

Wir werden später sehen, daß wir noch drei Konstanten benötigen. Wir wollen sie hier aufführen; sie sollen außerdem auf den Speicherplätzen ab 002 23C gespeichert vorliegen:

ØØ2 23C	ØØØØ ØØØØ
ØØ2 24Ø	ØØØØ ØØØ2
ØØ2 244	FØ

Wir hatten angenommen, daß unser Problemprogramm die Programmablaufsteuerung mit dem Programmstatuswort (PSW)

0000 0000 0000 2300

übertragen bekommt, so daß seine erste Instruktion auf dem Hauptspeicherplatz 002 300 beginnen muß. Da wir das Register 3 als Basisregister verwenden wollen, wenn wir uns

1) Wir müssen den Supervisorstatus voraussetzen, da wir mit privilegierten Instruktionen arbeiten wollen.

auf Instruktionen beziehen, müssen wir die Distanzadressen in den Instruktionen auf den Inhalt dieses Basisregisters, also auf 0000 2300, abstimmen. Zu Beginn unseres Programmes übertragen wir das Kanaladreßwort (CAW) des ersten Kanalprogrammes auf seinen erforderlichen Hauptspeicherplatz, auf 000 048 also, und löschen dann möglicherweise noch anstehende Unterbrechungen für das von uns eingesetzte EA-Gerät, d.h. für die IBM 2520 mit der EA-Adresse 00D. Wir vernachlässigen hierbei, daß durchaus auch die Möglichkeit besteht, daß eine hängende Unterbrechung für ein anderes EA-Gerät existiert. Wenn keine hängende Unterbrechung vorliegt, so liefert die Instruktion den Bedingungsschlüssel 0 ab. Weist die IBM 2520 dagegen eine hängende Unterbrechung auf, so setzt sie den Bedingungsschlüssel auf 1 und hebt die Unterbrechung auf. Damit wird bei der zweiten Ausführung der Instruktion TIO der Bedingungsschlüssel tatsächlich ebenfalls auf 0 gebracht. Die ersten Instruktionen unseres Problemprogrammes lauten also wie folgt:

Hauptspeicher-adresse	Distanz-adresse	Mnemonischer Operationsschlüssel	Instruktion
002 300	000	MVC	D2 03 0 048 2 030
002 306	006	TIO	9D 00 0 00D
002 30A	00A	BC	47 7 0 3 006
002 30E	00E	SIO	9C 00 0 00D
002 312	012	BC	47 3 0 3 006
002 316	016	BC	47 8 0 3 038

Wir dürfen hier noch einige Erläuterungen zu den BC-Instruktionen machen: Die erste BC-Instruktion verzweigt zurück zur TIO-Instruktion, wenn der BS [1] ungleich Null ist, die zweite verzweigt gleichfalls zurück zur TIO-Instruktion, jedoch dann, wenn der BS gleich 2 oder gleich 3 ist. Die letzte BC-Instruktion schließlich überspringt die nachfolgenden Instruktionen, sie verzweigt zur TIO-Instruktion auf dem Speicherplatz 002 340.

Nachdem jede Unterbrechung durch die TIO-Instruktion aufgehoben ist, geben wir die Instruktion SIO. Der BS muß 0 sein, daß während der wenigen Mikrosekunden, die die Ausführung dieser Instruktion dauert, irgendetwas mit dem EA-Gerät passiert ist; sicherheitshalber prüfen wir das nach: Ist der BS gleich 2 oder gleich 3, so verzweigen wir zurück zur TIO-Instruktion. (In der Praxis würden wir in eine Fehlerbehandlungsroutine hineingehen, wenn der BS auf 3 steht.) Der Fall, daß der BS gleich 1 ist, tritt ein, wenn die letzte Karte gelesen wurde. Wird für die IBM 2520 noch ein Befehl gegeben, nachdem die letzte Karte bereits gelesen wurde, so wird das Kanalprogramm nicht ausgeführt. Stattdessen werden die Statusbytes gespeichert, wobei die Bitstelle U_7 (Geräte-Ausnahmebedingung) auf 1 steht. Wir werden uns dafür entscheiden, den Programmteil zuerst zu schreiben, der sich mit der Behandlung der Bedingung „letzte Karte" befaßt. Deshalb verzweigen wir, wenn wir finden, daß der BS gleich 0 ist. Wird also nicht verzweigt, so wissen wir, daß der BS gleich 1 ist und daß entweder die Bedingung „letzte Karte" oder eine sonstige Ausnahmebedingung vorliegt. Für die Behandlung des Wertes 1 beim Bedingungsschlüssel setzen wir unser Problemprogramm mit folgenden Instruktionen fort:

1) BS ist die Abkürzung von Bedingungsschlüssel

Hauptspeicher-adresse	Distanz-adresse	Mnemonischer Operationsschlüssel	Instruktion
ØØ2 31A	Ø1A	MVC	D2 Ø Ø 2 Ø3F Ø Ø44
ØØ2 32Ø	Ø2Ø	L	58 5 Ø 2 Ø3C
ØØ2 324	Ø24	SR	1B 4 4
ØØ2 326	Ø26	D	5D 4 Ø 2 Ø4Ø
ØØ2 32A	Ø2A	LTR	12 4 4
ØØ2 32C	Ø2C	BC	47 8 Ø 3 ØØ6
ØØ2 33Ø	Ø3Ø	MVC	D2 Ø 3 Ø Ø48 2 Ø38
ØØ2 336	Ø36	SIO	9C Ø Ø Ø ØØD
ØØ2 33A	Ø3A	BC	47 7 Ø 3 Ø36
ØØ2 33E	Ø3E	BCR	Ø7 F 3

Diese Instruktionsfolge bedarf kurzer Erläuterungen. Durch die erste Instruktion wird das Gerätestatusbyte auf den Hauptspeicherplatz 002 23F übertragen. Durch die zweite Instruktion wird die Konstante von 002 23C ins Register 5 geladen; im rechtsbündigen Byte dieses Registers steht also das Gerätestatusbyte. Die dritte Instruktion löscht Register 4; Register 4 und 5 bilden damit den Dividenden für die nachfolgende Division durch 2 (4. Instruktion). Der als Ergebnis dieser Division im Register 4 auftretende Rest wird durch die Instruktion LTR auf Null geprüft. Ist der Rest gleich Null, d.h. ist die Geräte-Ausnahmebedingung (Bitstelle U_7 des Gerätestatusbytes) nicht vorhanden, so erfolgt durch die 6. Instruktion (BC) eine Verzweigung zurück zur TIO-Instruktion auf dem Hauptspeicherplatz 002 306.

Durch die 7. Instruktion wird das Kanaladreßwort für das 3. Kanalprogramm auf den Hauptspeicherplatz 000 048 übertragen, d.h. auf die zur Einleitung von Kanaloperationen erforderlichen Stellen. Die nachfolgende SIO-Instruktion startet das 3. Kanalprogramm, d.h. das Stanzen der Summe auf die letzte Karte. Führt die SIO-Instruktion auf einen von 0 verschiedenen Bedingungsschlüssel, so sorgt die 9. Instruktion (BC) dafür, daß eine Verzweigung zurück zur SIO-Instruktion auf dem Speicherplatz 002 336 erfolgt. Nach dem erfolgreichen Stanzen der letzten Karte kehren wir zur ersten Instruktion unseres Programmes zurück. Damit können wir die Verarbeitung eines neuen Kartenpaketes beginnen.

Wenn unsere erste SIO-Instruktion auf dem Hauptspeicherplatz 002 306 auf den Bedingungsschlüssel 0 geführt hat, so verzweigen wir bekanntlich durch die auf dem Speicherplatz 002 316 beginnende BC-Instruktion auf den Speicherplatz 002 340. Die auf diesem beginnende Instruktionsfolge sieht wie folgt aus:

Hauptspeicher-adresse	Distanz-adresse	Mnemonischer Operationsschlüssel	Instruktion
ØØ2 34Ø	Ø4Ø	TIO	9D Ø Ø Ø ØØD
ØØ2 344	Ø44	BC	47 7 Ø 3 Ø4Ø
ØØ2 348	Ø48	PACK	F2 2 3 1 Ø12 1 ØØØ
ØØ2 34E	Ø4E	PACK	F2 3 5 1 Ø15 1 ØØ4
ØØ2 35E	Ø54	AP	FA 3 2 1 Ø15 1 Ø12
ØØ2 35A	Ø5A	UNPK	F3 7 3 1 ØØA 1 Ø15
ØØ2 36Ø	Ø6Ø	MVZ	D3 Ø Ø 1 Ø11 2 Ø44
ØØ2 366	Ø66	MVC	D2 Ø 3 Ø Ø48 2 Ø34
ØØ2 36C	Ø6C	BC	47 F Ø 3 ØØ6

Dieser Programmteil sorgt für die geforderte Verarbeitung. Zwischen den beiden Instruktionen TIO und BC wird eine Schleife installiert, die erst verlassen wird, wenn der Bedingungsschlüssel gleich Null ist. Die 3. und 4. Instruktion dieser Instruktionsfolge packen den x-Wert und den y-Wert. Die Summe wird anschließend entpackt. Durch die MVZ-Instruktion wird der Zonenteil der Einerstelle der entpackten Dezimalzahl auf sedezimal F gesetzt, wodurch der Absolutbetrag der Summe ohne Vorzeichen erzeugt wird. Das Kanaladreßwort für das 2. Kanalprogramm wird durch die vorletzte Instruktion auf den Hauptspeicherplatz 000 048 gebracht, so daß mittels der nachfolgenden unbedingten Verzweigung, die auf den Speicherplatz 002 306 führt, dieses auch vorbereitet und gestartet werden kann (TIO und SIO).

Ziehen wir Fazit aus dem von uns verfaßten Problemprogramm: Die in der letzten Instruktionsfolge erscheinenden fünf Instruktionen mit den Distanzadressen 048, 04E, 054, 05A und 060 erledigen die gesamte in der Aufgabenstellung geforderte Verarbeitung. Die restlichen zwanzig Instruktionen haben wir für die Steuerung der EA eingesetzt. Für viele Computeranwendungen sind derartige Sachverhältnisse typisch. - Praktisch wird man die EA kaum selbst in ein Anwendungsprogramm aufnehmen, sondern stattdessen die Programmierunterstützung der IBM in Anspruch nehmen.

5.17. Kanalidentifizierung

Die Instruktion

Speicherung der Kanalidentifizierung —— Store Channel Identification	B2	STIDC

erfordert die Angabe des sedezimalen Ziffernpaares 03 im zweiten Byte, d.h. auf den Bitstellen 8 bis 15. Alle acht Bits im zweiten Byte werden nämlich bei der Instruktionsausführung entschlüsselt. Wie alle in diesem Kapitel behandelten Instruktionen ist auch die Instruktion STIDC eine privilegierte Instruktion im S-Format. Die für diese Instruktion gebildete Adresse bezieht sich ausschließlich auf den Kanal, die Geräteadresse wird ignoriert.

Wenn eine gefüllte Anzeigeleiste im Kanal vorliegt, so wird der Bedingungsschlüssel auf 1 gesetzt. Allen Bitstellen des CSW mit Ausnahme von A (Bitstelle 5) und von S_5 (Bitstelle 5 im Kanalstatusbyte), denen eine 1 zugewiesen wird, werden auf Null gesetzt. Danach wird die Instruktionsausführung abgeschlossen. Ereignet sich ein Kanalsteuerfehler während der Ausführung der Instruktion, so wird der BS [1] auf 1 gesetzt und den beiden Statusbytes der sedezimale Wert 0004 zugewiesen, d.h. die Bitstelle S_5 im Kanalstatusbyte wird auf 1 gesetzt. Danach wird die Operation beendet. Ist der Kanal nicht arbeitsfähig (betriebsbereit), so wird dem BS der Wert 3 zugewiesen. Ist dagegen der Kanal beschäftigt oder enthält er eine hängende Unterbrechung, so wird der BS auf 2 gesetzt, falls der Zustand des Kanals die reguläre Ausführung dieser Instruktion verhindert. Normalerweise wird der BS auf 0 gesetzt und vier Bytes (gleich acht Sedezimalziffern) werden auf die festen Hauptspeicherplätze mit den Adressen 168 bis 171 (sedezimal 0000A8 bis 0000AB) gespeichert. Die erste weggespeicherte Sedezimalziffer dient zur Identifizierung des Kanaltyps:

1) BS ist in diesem Abschnitt die Abkürzung von Bedingungsschlüssel.

Sedezimalziffer	Kanaltyp
0	Selektorkanal
1	Bytemultiplexkanal
2	Blockmultiplexkanal

Die nächsten drei Sedezimalziffern geben die Modellnummer des betreffenden Kanales an. Diese wiederum hängt entscheidend vom benutzten Modell der Computerfamilie IBM /370 ab. Die letzten vier Sedezimalziffern geben schließlich die Maximallänge der Kanalanzeigeleiste an. Dieser Wert ist gleich Null bei denjenigen Kanälen, die keine Anzeigeleiste bei Fehlern generieren (siehe hierzu auch Abschnitt 5.4.).

Diese Instruktion wird hauptsächlich dazu benutzt, um genügend Hauptspeicherplatz für die Aufnahme einer Kanalanzeigeleiste bereitzustellen.

Aufgabengruppe 12

1) Jedes der folgenden Kanalprogramme enthält einen Fehler. In jedem Fall ist vorausgesetzt, daß das Kanalprogramm für eine IBM 2520 bestimmt ist und daß das erste CCW auf dem Speicherplatz 001 800 steht.

a)
```
01 001000
A0 000010
11 001307
A0 000041
C2 0013A4
20 000008
```

b)
```
C8 001000
20 000050
```

c)
```
C0 001000
20 000050
```

d)
```
01 001000
40 000050
C2 001000
40 000050
08 001804
00 000000
```

e)
```
11 001000
80 000024
08 001810
00 000038
00 001302
40 000018
C2 001000
00 000050
```

2) Vor Einleitung der Kanalprogramme (Aufgabe 1) sind TIO-Instruktionen gegeben worden, die einen Bedingungsschlüssel von 0 ergaben. Danach werden SIO-Instruktionen zur Einleitung der Kanalprogramme gegeben. Welches Kanalprogramm bzw. welche Kanalprogramme führen dabei auf einen Bedingungsschlüssel 0?

3) Welche Informationen werden bei Kanalende durch die hier bei auftretende Unterbrechung gespeichert, und welche Maßnahmen wurden durchgeführt, wenn jedes der Programme von Aufgabe 1) beendet wird, sofern es überhaupt begonnen wurde?

4) Üblicherweise werden bei der Behandlung von EA-Vorgängen bei der IBM /370 alle Unterbrechungen zugelassen. Jedem EA-Gerät wird deshalb sinnvollerweise ein Byte zugeordnet, in denen wichtige Informationen über den gegenwärtigen Zustand des Gerätes festgehalten werden.

Angenommen, wir wollen ein Programm schreiben, das ein Kartenpaket von der IBM 2520 liest, und wir wollen die Summe derjenigen Zahlen ermitteln, die auf den Spalten 1 bis 5 jeder Lochkarte abgelocht sind. Nach dem Einlesen der letzten Karte soll diese Summe auf einen Zeilendrucker gedruckt werden. Wir sehen für die IBM 2520 ein Byte vor, in das wir die relevanten Informationen über dieses Gerät einspeichern. Eine Routine zur Behandlung von Unterbrechungen soll die erste Bitstelle dieses Bytes auf 0 setzen, wenn Geräteende festgestellt wird. Die gleiche Routine soll weiterhin der sechsten Bitstelle dieses Bytes eine 1 zuweisen, wenn

entweder auf der Bitstelle U_6 des Gerätestatusbytes oder auf den Bitstellen S_2 bis S_7 des Kanalstatusbytes eine 1 vorhanden ist; mit anderen Worten also, wenn irgendein Fehler aufgetreten ist. Ferner soll noch die siebente Bitstelle auf 1 gesetzt werden, wenn die Bitstelle U_7 des Gerätestatusbytes (Geräte-Ausnahmebedingung) eine 1 enthält. Diese Ausnahmebedingung entsteht durch das Einlesen der letzten Lochkarte.

Im Programm selbst sind zwei separate Einlesegebiete vorzusehen, so daß die Berechnung während des Einlesens der nächsten Lochkarte fortgesetzt werden kann.

Schreibe nun folgende Einzelprogramme:

a) Routine zur Behandlung von Unterbrechungen. Diese soll die Ablaufsteuerung zum ursprünglichen Programm zurückgeben. Die erste Instruktion dieser Routine soll auf Speicherplatz 002 000 beginnen. Es ist der Basissteuerungs-Modus anzunehmen.

b) Eine Routine, die die Informationen von der nächsten Karte in das Eingabegebiet einliest, dessen Adresse im Register 15 steht. Diese Routine soll ab Speicherplatz 002 400 gespeichert werden.

c) Ab Speicherplatz 002 800 soll eine Routine gespeichert werden, die die für die Berechnung notwendigen Daten packt usw. und die die anderen Routinen aufruft, falls das erforderlich ist.

d) Auf den Speicherplätzen ab 002 C00 soll die Dateiende-Routine gespeichert werden, die Routine also, die die Summe ausgibt, d.h. druckt. Für den Zeilendrucker ist die EA-Adresse 00E anzunehmen und der Befehlsschlüssel 01 zu benutzen.

Um dem Programm die Ablaufsteuerung zurückgeben zu können, das beim Auftreten einer Unterbrechung gerade im Ablauf begriffen war, soll die Instruktion LPSW

```
82 00 0 038
```

verwendet werden, d.h. Laden des Programmstatuswortes vom Hauptspeicherplatz mit der sedezimalen Adresse 000 038 .

6. Eingabe/Ausgabe-Geräte (EA-Geräte)

In diesem Kapitel wollen wir die Arbeitsweise einiger EA-Geräte behandeln, die häufig an das System IBM /370 angeschlossen werden.

6.1. Das Interface (Schnittstellen)

Bei jedem Unterkanal läuft eine Auswahlleitung von der Ausgangsanschlußstelle des Unterkanals zur Steuereinheit mit der höchsten Priorität, die am Unterkanal angeschlossen ist. Von dort geht sie zur Steuereinheit mit der zweithöchsten Priorität usw.; sie erreicht schließlich die Steuereinheit mit der niedrigsten Priorität. Von dieser führt die Leitung zurück zur Eingangsbuchse am Unterkanal. Neben dieser Auswahlleitung gibt es mehrere andere Verbindungen, die von EA-Gerät zu EA-Gerät geführt sind. Sie können wie folgt klassifiziert werden:

a) Ausgangsübertragungswege (Ausgangsleitungen)

Diese aus 9 Leitungen (8 für Daten, 1 für die Prüfung) bestehenden Verbindungen übertragen die Befehle, die Geräteadressen und die Daten vom Unterkanal zum Gerät.

b) Eingangsübertragungswege (Eingangsleitungen)

Diese ebenfalls aus 9 Leitungen bestehenden Verbindungen übertragen Geräteadressen, Statusinformationen und Daten von den EA-Geräten zum Unterkanal.

c) Kennungsleitungen

Die auf diesen Leitungen übertragenen Impulse identifizieren die auf den Übertragungswegen ausgetauschten Informationen als Befehle, Daten usw.

d) Meßleitungen

Diese Leitungen schalten die Meßeinrichtungen (Meßgeräte), die den Gebrauch (die Benutzungszeit) registrieren, ein und aus. Die Meßeinrichtungen dienen zur Bestimmung der Stundenzahl, die das betreffende Gerät während eines Monats benutzt wurde.

Die Aktivität eines Kanals kann durch ein EA-Gerät oder durch den Kanal selbst eingeleitet werden. Der Kanal leitet die Aktivität als Ergebnis einer gegebenen SIO- oder einer anderen EA-Instruktion ein. Die Geräteadresse wird auf den Ausgangsübertragungsweg geschickt; ebenso wird auch ein Impuls auf die Kennungsleitung für Adreßausgänge gegeben. Wenn dieser Adreßausgangsimpuls der Reihe nach von den einzelnen Geräten empfangen wird, überprüfen diese die auf dem Ausgangsübertragungsweg ankommende Adresse. Erkennt ein Gerät diese Adresse als seine eigene, so erregt es die Kennungsleitung für Adreßeingänge; außerdem sendet es seine eigene Adresse über den Eingangsübertragungsweg. Bestätigt kein Gerät durch einen solchen Gegenimpuls den Empfang der ausgesandten Adresse, so kommt der Kanal zu dem Schluß, daß das betreffende EA-Gerät nicht arbeitsfähig (betriebsbereit) sein muß. Neben der Anwendung des Impulses zur Bestätigung der aufgenommenen Adresse erregt das Gerät auch eine andere Leitung, damit die Steuereinheit mit dem Unterkanal unter Ausschluß aller anderen Geräte verbunden wird. Empfängt nun der Kanal den Impuls, der die Ankunft der Adresse bestätigt, so schickt er einen aus lauter Nullen bestehenden Testbefehl auf den Ausgangsübertragungsweg und erregt die Kennungsleitung für Befehlsausgänge. Das Gerät reagiert darauf mit dem Senden seines Gerätestatusbytes auf dem Eingangsübertragungsweg und löscht anschließend sein Gerätestatusbyte. Danach wird der Befehl des ersten Kanalbefehlswortes (CCW) auf den Ausgangsübertragungsweg geschickt und erneut die Kennungslei-

tung für Befehlsausgänge erregt. Anschließend erfolgt die Datenübertragung. Enthält die Bitstelle niedrigster Ordnung des Befehlsschlüssels, d.h. die am weitesten rechts gelegene Bitstelle, eine 1, so werden die Daten vom Kanal zum Gerät transportiert, im anderen Falle vom Gerät zum Kanal.

Die Bitstelle für Kanaldatenfehler (S_4 im Kanalstatusbyte) wird immer dann auf 1 gesetzt, wenn auf den Eingangsübertragungswegen bei einem Byte eine falsche Parität, d.h. eine gerade Anzahl von Bitstellen, die eine 1 enthalten, entdeckt wird. Die Bitstelle für Kanalsteuerfehler (S_5) wird dann auf 1 gesetzt, wenn ungültige Signale auf den Leitungen entdeckt werden, die den Kanal mit den Steuereinheiten verbinden. Es wird erwartet, daß die Geräte innerhalb einer vorgegebenen Frist auf ein empfangenes Signal reagieren. Erfolgt während dieser Zeit keine Reaktion, so wird ein Interface-Steuerfehler auf der Bitstelle S_6 angezeigt.

Die Aktivität kann auch durch das Gerät selbst eingeleitet werden. Immer, wenn ein Kanal frei, d.h. unbeschäftigt ist, ruft er seine angeschlossenen Geräte der Reihe nach auf. Das erste Gerät, das ein Byte im Mehrpunktbetrieb senden will, schickt seine Adresse auf den Eingangsübertragungsweg und erregt außerdem die Kennungsleitung für Adreßeingänge. Alle anderen Vorgänge laufen dann genau so ab, als ob der Unterkanal die Aktivität eingeleitet hätte. Nach einem kompletten Rundruf aller angeschlossenen EA-Geräte, bei dem kein Gerät Daten senden wollte, werden die Geräte erneut nacheinander aufgerufen. Dieser gestattet dann den Geräten mit einer Unterbrechungsbedingung, wie z.B. „Geräteende", den Kanal zu benutzen.

Die Geräte unterscheiden sich in ihren Reaktionen, wenn der Kanal sie nicht schnell genug bedienen kann. Bei einigen Geräten, wie z.B. dem Kartenleser IBM 2501, gehen Daten verloren, wenn der Unterkanal sie nicht in dem Augenblick annehmen kann, wenn sie zur Übertragung bereit stehen. Das liegt daran, daß ihre Geschwindigkeit nicht mehr geändert werden kann, wenn bei ihnen die mechanische Bewegung erst einmal begonnen hat. Andere Geräte dagegen, wie z.B. der Drucker IBM 3211, halten einfach an und warten auf ihre Bedienung durch den Kanal solange, wie es notwendig ist; übermäßige Verzögerungen können wohl die Druckgeschwindigkeit herabsetzen, doch gehen keine Daten verloren. Die Bedingung, daß Daten verloren gehen, bezeichnet man gewöhnlich als Datenverlustbedingung [1]. Wenn die Datenverlustbedingung beim ersten Datenbyte nach der Befehlskettung auftritt, so wird der Bitstelle für Kettungsfehler S_7 im Kanalstatusbyte eine 1 zugewiesen. In allen anderen Fällen wird der Bitstelle für Datenverlust (Bitstelle 5) im ersten Geräteprüfbyte (siehe hierzu Abschnitt 5.6.) eine 1 zugewiesen. Bei der Festlegung von Anschlußprioritäten für EA-Geräte sollte die höchste Priorität denjenigen Geräten zugeordnet werden, bei denen eine Datenverlustbedingung entstehen kann.

Jedes EA-Gerät besitzt mindestens ein Geräteprüfbyte, jedoch können einzelne Gerätetypen bis zu 20 besitzen. Die Geräteprüfbytes haben bei den verschiedenen Geräten unterschiedliche Bedeutung. Eine Ausnahme bilden hier nur die Bitstellen 0 bis 5 des ersten Prüfbytes, denen stets die gleiche Bedeutung zukommt.

Das Vorhandensein einer anomalen 1 auf irgendeiner Bitstelle irgendeines Prüfbytes bewirkt, daß die Bitstelle für Gerätefehler (U_6) im Gerätestatusbyte ebenfalls auf 1 gesetzt wird. Die Prüfbitstellen werden im allgemeinen zurückgesetzt, wenn es zur Ausführung des nächsten Kanalbefehles kommt, der nicht gleich einem der Befehle „Prüfen" oder „NOP" ist. Eine Ausnahme von dieser Regel bildet jedoch die Bitstelle 1 des ersten Prüfbytes (Eingriff erforderlich). Bedingungen wie a) Kartenleser nicht be-

1) Im angloamerikanischen Sprachraum als „overrun" bezeichnet.

reit, b) Papierende beim Drucker usw. werden erst dann aufgehoben, wenn die Ursache beseitigt ist. Der Prüfbitstelle 0 des ersten Prüfbytes kommt die Bedeutung der „Befehlszurückweisung" zu. Ihr wird stets dann eine 1 zugewiesen, wenn der gegebene Befehl vom EA-Gerät nicht ausgeführt werden kann. Eine Befehlszurückweisung kann auf den folgenden Ursachen beruhen:

a) Das EA-Gerät ist nicht für die Ausführung des betreffenden Befehles entworfen worden,
b) der Befehl erfordert zu seiner Ausführung eine zusätzliche, jedoch nicht vorhandene Geräteausrüstung,
c) das Gerät befindet sich gegenwärtig in einem Zustand, der die Ausführung eines Befehles ausschließt, der unter anderen Umständen jedoch ausgeführt werden kann.

Der Bitstelle 2 im ersten Prüfbyte kommt die Bedeutung der Anzeige eines Fehlers zu, der auf dem Ausgangsübertragungsweg entstanden ist (Übertragung eines Zeichens mit einer geraden Anzahl von Einerbits). Der Bitstelle 3 des ersten Prüfbytes (Ausrüstungsprüfung) wird eine 1 zugewiesen, wenn das EA-Gerät eine mechanische oder elektrische Störung aufweist (Fehler in der Maschinenausrüstung). Der Bitstelle 4 des ersten Prüfbytes (Datenprüfung) wird schließlich dann eine 1 zugewiesen, wenn ein Fehler auf dem externen Medium (Datenträger) festgestellt wird. Damit ist nicht ein Gerätefehler gemeint, sondern vielmehr ein Fehler, der durch eine schlecht gestanzte Karte, ein verbrauchtes Magnetband usw. verursacht wird.

Die Zuweisung von Einerbits zu den Prüfstellen für Datenprüfung bzw. für Datenverlust (Bitstellen 4 bzw. 5) im ersten Prüfbyte hat zur Folge, daß es unbestimmt ist, ob die Datenübertragung fortgesetzt wird oder nicht. Bei den meisten anderen Prüfbedingungen wird hingegen die weitere Datenübertragung sofort unterbunden.

6.2. Der Kartenleser IBM 2501

Die Abb. 6.1. zeigt den Kartenleser IBM 2501. Die Karten laufen von einem Kartenmagazin zu einer vor der Lesestation liegenden Stelle, passieren dann eine Lesestation, die mit 12 Photozellen ausgerüstet ist und werden anschließend in einem Ablagefach abgelegt.

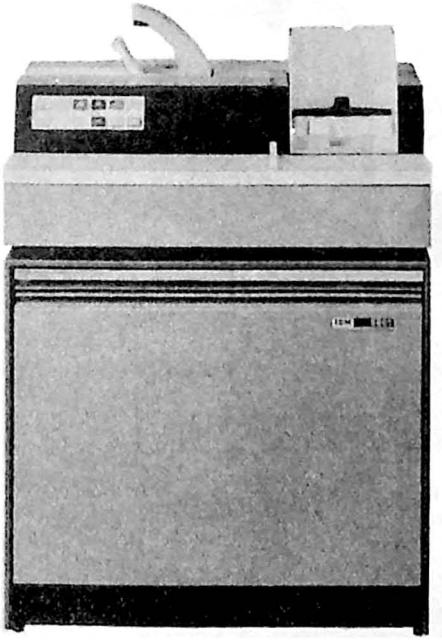


Abb. 6.1. Kartenleser IBM 2501

(Lochungen in den Karten werden dadurch ermittelt, daß das ausgestrahlte Licht, durch die Löcher hindurchgehend, von den Photozellen empfangen wird.) Die Kartenbahn ist in der Abb. 6.2. dargestellt. Das Modell B1 liest die Karten mit einer maximalen Geschwindigkeit von 600 Karten pro Minute; das Modell B2 bringt es auf eine Geschwindigkeit von 1000 Karten pro Minute.

Das Lesen kann in zwei verschiedenen Betriebsarten (Modi) erfolgen. Bei der Betriebsart 1 (Datenmodus 1) wird jede gelesene Kartenspalte sofort in den EBCDI-Code übersetzt, und das sich daraus ergebende Byte wird zum Kanal gesendet. Bei der Betriebsart 2 (Datenmodus 2) findet keine Übersetzung statt; der Einbau der dafür erforderlichen Ausrüstung kann nur zu Sonderkosten erfolgen. Der Inhalt jeder Kartenspalte geht vielmehr wie folgt in zwei Bytes hinein: Die Zeilen 12 bis 3 ergeben die Bitstellen 2 bis 7 des ersten Bytes, die Zeilen 4 bis 9 die Bitstellen 2 bis 7 des zweiten Bytes. Die Bitstellen 0 und 1 der beiden Bytes werden auf 0 gesetzt. Bei der Betriebsart 1 wird somit der Buchstabe A zu C1 (sedezimal) in einem Hauptspeicherbyte gespeichert, bei der Betriebsart 2 geht er in zwei Hauptspeicherbytes, in denen dann also 2400 (sed) gespeichert ist. Bei der Betriebsart 2 werden alle Lochkombinationen als gültig angesehen; bei der Betriebsart 1 hingegen wird der Inhalt von Kartenspalten, bei denen mehr als eine Lochung in den Zeilen 1 bis 7 vorhanden ist, als ungültig angesehen und verursacht infolgedessen einen Datenfehler.

Beim Kartenleser IBM 2501 werden vier verschiedene Befehlsschlüssel verwendet:

02	Lesen in der Betriebsart 1 (Read Data Mode 1)
22	Lesen in der Betriebsart 2 (Read Data Mode 2)
03	Keine Operation (Nop)
04	Prüfen (Sense)

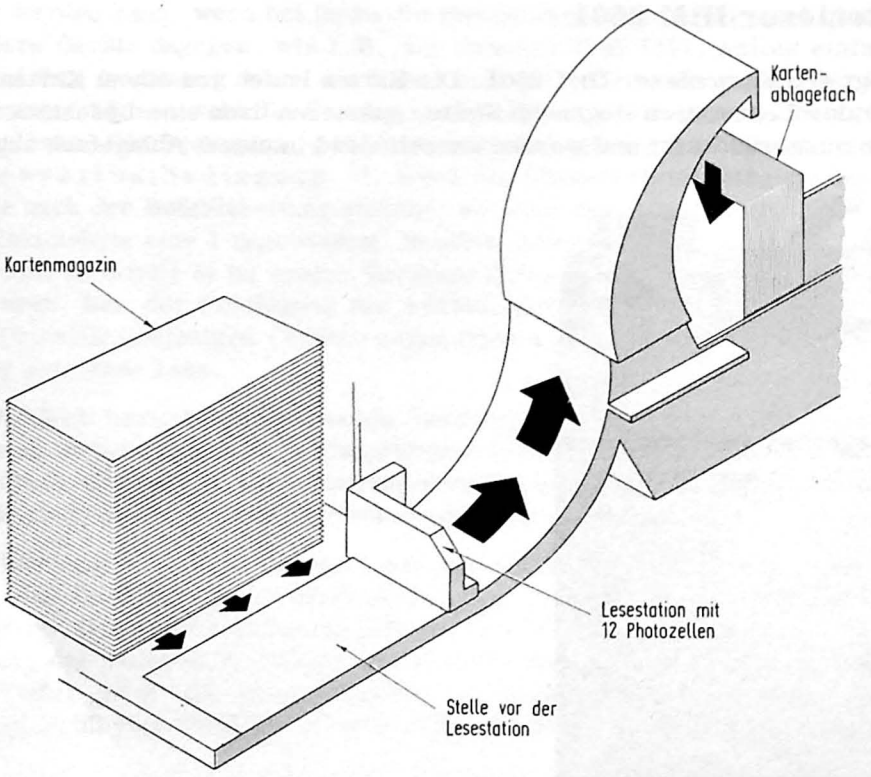


Abb. 6.2. Die Kartenbahn des Kartenlesers IBM 2501

Die Bitstellen des Gerätestatusbytes besitzen folgende Bedeutung:

a) Die Bitstellen 0, 1 und 2 werden nicht verwendet.

b) Bitstelle 3 — In Arbeit

c) Bitstelle 4 — Kanalende
Die Kanalendebedingung tritt auf, sobald das letzte Byte zum Kanal übertragen worden ist. Wenn weniger als 80 Spalten gelesen werden, so kommt es zu dieser Bedingung in dem Augenblick, in dem das Zählerfeld des CCW (des Kanalbefehlswortes) auf 0 abgebaut ist.

d) Bitstelle 5 — Geräteende
Diese Bedingung tritt dann auf, wenn die Karte vollständig die Lesestation passiert hat. Zu dieser Bedingung kommt es wenigstens 3, 5 Millisekunden nach Feststellung der Kanalendebedingung.

e) Bitstelle 6 — Gerätefehler
Diese Bitstelle wird auf 1 gesetzt, wenn irgendeine Bitstelle des Prüfbytes auf 1 steht.

f) Bitstelle 7 — Geräteausnahmebedingung
Beim ersten Leseversuch, der nach dem Lesen der letzten Karte vorgenommen wird, kommt es zur Geräteausnahmebedingung (während der Selektionszeit). Dadurch wird der Bedingungsschlüssel für die SIO-Instruktion auf 1 gesetzt.

Die einzelnen Bitstellen des Prüfbytes haben folgende Bedeutung:

a) Bitstelle 0 — Befehlszurückweisung

b) Bitstelle 1 — Eingriff erforderlich
Ein manueller Eingriff ist unter folgenden Umständen erforderlich: Volles Kartenablagefach; Kartenanstoß in der Kartenbahn; leeres Kartenmagazin, ohne daß die Taste „LETZTE KARTE" gedrückt worden ist; nicht betriebsbereit; Fehler bei den Photozellen.

c) Bitstelle 2 — Fehler in der Ausgangsleitung (im Ausgangsübertragungsweg)

d) Bitstelle 3 — Ausrüstungsfehler

e) Bitstelle 4 — Datenfehler

f) Bitstelle 5 — Datenverlust

g) Die Bitstellen 6 und 7 haben keine Bedeutung.

Die Abb. 6.3. zeigt die Bedienungs- und Kontrollorgane der IBM 2501.

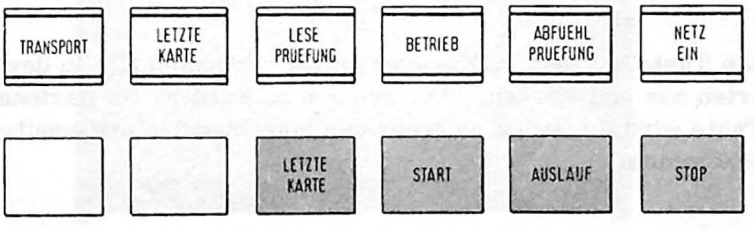


Abb. 6.3. Bedienungs- und Kontrollorgane der IBM 2501

Die sechs Anzeigelampen haben folgende Bedeutung:

a) NETZ EIN
Die IBM 2501 ist mit elektrischer Spannung versorgt.

b) ABFUEHLPRUEFUNG
Durch diese Anzeigelampe wird ein Kartenanstoß in der Kartenbahn oder ein Schaden (Ausfall) der Photozellen signalisiert. Durch diese Ereignisse wird gleichzeitig die Bitstelle 1 des Prüfbytes auf 1 gesetzt.

c) BETRIEB
Diese Anzeigelampe leuchtet bei Erfüllung der folgenden Bedingungen auf:
(1) Das Kartengerät ist mit Spannung versorgt.
(2) Eine Karte befindet sich an der Stelle vor der Lesestation.
(3) Entweder leuchtet die Lampe LETZTE KARTE auf, oder es befinden sich Karten im Kartenmagazin.
(4) Das Ablagefach ist nicht voll.
(5) Die Lampe ABFUEHLPRUEFUNG leuchtet nicht auf.
(6) Die Verkleidungen des Kartengerätes sind nicht geöffnet.
(7) Die Taste STOP wurde seit dem letzten Drücken der Taste Start nicht gedrückt.
Wenn die Lampe BETRIEB nicht aufleuchtet, ist gleichzeitig die Bitstelle 1 des Prüfbytes auf 1 gesetzt.

d) LESEPRUEFUNG
Die Bitstelle 3 des Prüfbytes enthält eine 1.

e) LETZTE KARTE
Wenn die letzte Karte vom Kartenmagazin in die Kartenbahn eingezogen ist, hält der Kartenleser an. Damit nun die letzte Karte gelesen werden kann, muß die Taste LETZTE KARTE gedrückt werden. Als Folge davon leuchtet die Anzeigelampe LETZTE KARTE auf, wodurch nunmehr das Lesen der letzten Karte ermöglicht wird. Nach dem Ablegen der letzten Karte wird diese Lampe ausgeschaltet.

f) TRANSPORT
Diese der Gültigkeitsprüfung dienende Anzeigelampe zeigt durch ihr Aufleuchten an, daß die Bitstelle 4 des Prüfbytes eine 1 enthält.

Die vier Tasten haben folgende Bedeutung:

a) START
Das Drücken dieser Taste bewirkt, daß die erste Karte in die Kartenbahn der IBM 2501 eingezogen wird. Wird die Taste nach dem Stoppen der Maschine erneut gedrückt, so wird der Kartenleser wieder in Betrieb gesetzt (die Lampe BEREIT leuchtet auf).

b) AUSLAUF
Drückt man diese Taste bei leerem Kartenmagazin, so laufen alle in der Kartenbahn befindlichen Karten aus und werden, ohne gelesen zu werden, im Kartenablagefach abgelegt. Diese Taste wird meist bei auftretenden Maschinenfehlern benutzt, um die Kartenbahn freizubekommen.

c) STOP
Das Niederdrücken dieser Taste versetzt die IBM 2501 in den nicht betriebsbereiten Zustand und hält die Maschine nach Abschluß der laufenden Operation an.

d) LETZTE KARTE
Damit die letzte Karte gelesen werden kann, muß diese Taste gedrückt werden. Dadurch wird gleichzeitig auch die Anzeigelampe LETZTE KARTE eingeschaltet.

Zur Bedienung des Kartenlesers sind in der Regel folgende Handgriffe erforderlich:

a) Einlegen der Karten in das Kartenmagazin
b) Drücken der Taste START
c) Drücken der Taste LETZTE KARTE, nachdem die letzten zu lesenden Karten ins Kartenmagazin eingelegt wurden.

Tritt beim Kartenleser eine besondere Bedingung auf, so entfernt man die noch im Kartenmagazin liegenden Karten und drückt die Taste AUSLAUF. Danach entnimmt man dem Ablagefach die letzte abgelegte Karte (diese wurde noch nicht gelesen), legt sie vor das dem Magazin entnommene Kartenpaket und packt die gesamten Karten erneut in das Magazin. Nach einem Fehler kann die Lampe LETZTE KARTE verlöscht sein; deshalb muß die Wiederanlaufprozedur erforderlichenfalls auch das Drücken der Taste LETZTE KARTE mit einschließen.

6.3. Das Kartengerät IBM 2520

Wir sind dem Kartengerät IBM 2520 bereits im Kapitel 5 begegnet. Auch hier werden wie bei der IBM 2501 die Karten von einem Kartenmagazin zugeführt und passieren eine mit Photozellen ausgestattete Lesestation. Zur gleichen Zeit wird jedoch die Karte, die der zu lesenden Karte vorausgeht, unter einer mit 80 Stanzstempeln ausgerüsteten Stanzstation hindurchgeführt und anschließend in eines von zwei Ablagefächern abgelegt (siehe auch Abb. 5.2.). Das Modell B1 liest und stanzt Karten mit einer Geschwindigkeit von 500 cd/m [1]. Das Modell B2 ist identisch dem Modell B1, doch fehlt bei ihm die Lesestation. Das Modell B3 stanzt Karten mit einer Geschwindigkeit von 300 cd/m; auch bei ihm fehlt die Lesestation. Die Karten durchlaufen die Modelle 2 und 3 so, als ob bei ihnen eine Lesestation vorhanden wäre; ein Lesen findet allerdings nicht statt.

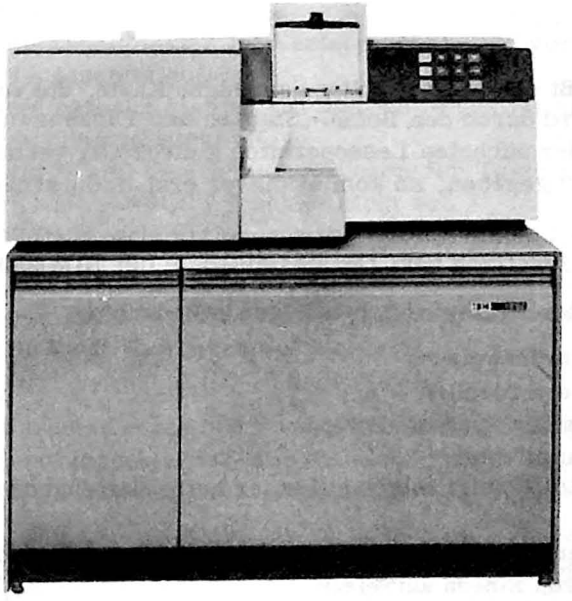


Abb. 6.4. Das Kartengerät IBM 2520

1) cd/m ist die gebräuchliche Maßeinheit für die Leistungsfähigkeit von Lochkartengeräten; sie kommt aus dem Englischen (cards/minute) und bedeutet demzufolge „Karten pro Minute".

Die von der IBM 2520 verwendeten Befehlsschlüssel sind in der Abb. 6.5. dargestellt. Der Befehl „Stanzen, Zuführen und Fachauswahl" überträgt die Daten zum Stanzpuffer, führt eine Karte zu und stanzt diese. Die Karte jedoch, die zur gleichen Zeit die Lesestation passiert, wird nicht gelesen. Beim Modell B1 ist einigen Befehlen ein gewisser Typ zugeordnet, nämlich AA, AB, BA und BB. Es ist notwendig, daß auf jeden Befehl ein solcher nachfolgt, dessen Befehlstyp mit dem Buchstaben beginnt, mit dem der Befehlstyp des vorhergehenden Befehles aufhört. Zum Beispiel kann im Anschluß an einen AB-Befehl ein BA- oder ein BB-Befehl gegeben werden, doch niemals ein AA- oder ein anderer AB-Befehl. Beim Zusammenstellen einer Befehlsfolge sind Nop- und Prüfbefehle auszuklammern. Der erste Befehl nach dem Starten des Gerätes muß immer ein AA- oder ein AB-Befehl sein, damit die erste Karte hinter die Lesestation transportiert werden kann. Die Betriebsart 2 steht für die IBM 2520 ebenfalls zur Verfügung, auch hier allerdings nur zu Sonderkosten.

Betriebsart		1		2	
Ablagefach		1	2	1	2
Stanzen, Zuführen und Fachauswahl	BB	01	41	21	61
Stanzen und Fachauswahl *)	BA	09	49	29	69
Fachauswahl *)	BA	23	63	23	63
Lesen, Zuführen und Fachauswahl *)	AA	02	42	22	62
Lesen und Zuführen *)	AB	C2		E2	
Keine Operation (Nop)	—	03			
Prüfen (Sense)	—	04			

*) Gilt nur für das Modell B1

Abb. 6.5. Die Befehlsschlüssel bei der IBM 2520

Die Fachauswahl beeinflußt den weiteren Weg derjenigen Karte, die vor der Stanzstation liegt. Beim Modell B1 wird durch den Befehl „Stanzen und Fachauswahl" der Stanzpuffer geladen, damit während der nächsten Leseoperation gleichzeitig gestanzt werden kann. Wird der Befehl „Fachauswahl" gegeben, so kommt dieser erst beim nächsten Lesebefehl zum Tragen.

Das Gerätestatusbyte ist identisch dem Gerätestatusbyte der IBM 2501.

Die einzelnen Bitstellen des Prüfbytes haben folgende Bedeutung:

a) Bitstelle 0 — Befehlszurückweisung
b) Bitstelle 1 — Eingriff erforderlich
c) Bitstelle 2 — Fehler in der Ausgangsleitung
d) Bitstelle 3 — Ausrüstungsfehler
   Eine 1 auf dieser Bitstelle zeigt folgende Fehler beim Kartengerät an:
   (1) Stanzfehler
   (2) Unkorrekte Parität im Stanzpuffer, d.h. im Stanzpuffer steht ein Zeichen, dessen Bitstellen eine gerade Anzahl von Einsen aufweisen.
   (3) Während der Betriebsart 1 wurde ein ungültiges Zeichen gestanzt.
   (4) Eine Kartenspalte wurde nicht richtig gelesen (gilt nur für das Modell B1).
e) Bitstelle 4 — Datenfehler
   Beim Lesen wurde eine Karte festgestellt, die auf einer Spalte mehr als eine Lochung in den Zeilen 1 bis 7 aufweist (gilt daher nur für das Modell B1).

f) Bitstelle 5 — Datenverlust
Gilt nur für das Modell B1.

g) Die Bitstellen 6 und 7 haben keine Bedeutung.

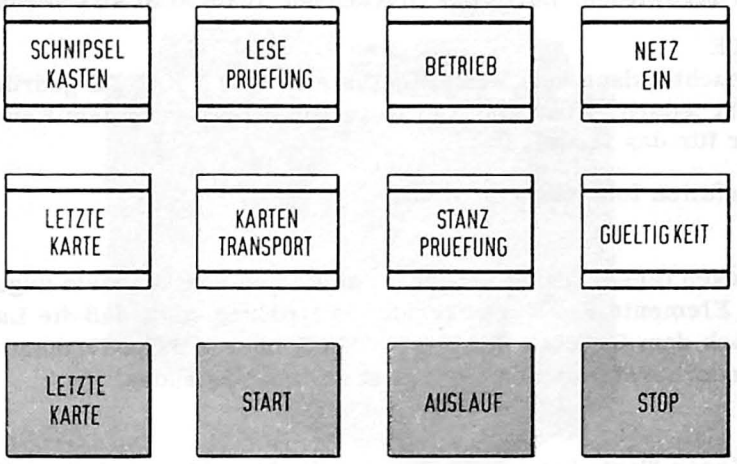


Abb. 6.6. Bedienungs- und Kontrollorgane der IBM 2520

In der Abb. 6.6. sind die Bedienungs- und Kontrollorgane des Kartengerätes abgebildet.

Es sind folgende acht Anzeigelampen vorhanden:

a) NETZ EIN

b) BETRIEB
Diese Lampe leuchtet auf, wenn die folgenden Bedingungen erfüllt sind:
(1) Das Kartengerät ist mit Spannung versorgt.
(2) Die Ablagefächer sind nicht gefüllt.
(3) Es befinden sich entweder Karten im Kartenmagazin und vor der Lesestation oder die Lampe LETZTE KARTE leuchtet auf.
(4) Der Schnipselkasten befindet sich an Ort und Stelle und ist nicht voll.
(5) Keine Maschinenverkleidung steht offen.
(6) Die Taste STOP wurde seit dem letzten Drücken der Taste START nicht gedrückt.
(7) Die Lampe KARTENTRANSPORT leuchtet nicht auf.
Wenn die Lampe BETRIEB nicht aufleuchtet, so ist ein manueller Bedienereingriff notwendig.

c) LESEPRUEFUNG
Diese Lampe erfüllt nur beim Modell B1 ihre Funktion. Gleichzeitig mit ihrem Aufleuchten wird auch die Bitstelle 3 des Prüfbytes auf 1 gesetzt.

d) SCHNIPSELKASTEN
Im Schnipselkasten werden die aus den Karten gestanzten Schnipsel aufgefangen. Wenn der Kasten voll oder gar nicht eingesetzt ist, so geht diese Lampe an. Nach dem Entleeren des Kastens und seinem Einschieben verlöscht die Lampe.

e) GUELTIGKEIT
Diese Lampe leuchtet dann auf, wenn die Bitstelle 4 des Prüfbytes eine 1 enthält. Gilt nur für das Modell B1.

f) STANZPRUEFUNG
Das Aufleuchten dieser Lampe zeigt an, daß eine Karte unkorrekt gestanzt wurde. Die Bitstelle 3 des Prüfbytes wird gleichzeitig auf 1 gesetzt. Die falsch gestanzte Karte wird grundsätzlich ins Ablagefach 1 transportiert.

g) KARTENTRANSPORT
Die in der Kartenbahn befindlichen Karten sind nicht einwandfrei weitertransportiert worden. Mit dem Aufleuchten dieser Lampe wird gleichzeitig auch der Bitstelle 1 (Eingriff erforderlich) des Prüfbytes eine 1 zugewiesen. Durch das Drücken der Taste AUSLAUF verlöscht diese Lampe.

h) LETZTE KARTE
Diese Lampe leuchtet dann auf, wenn die Taste LETZTE KARTE gedrückt wurde. Die Lampe verlöscht jedoch, wenn eine Geräteausnahmebedingung dem Kanal signalisiert wurde. Gilt nur für das Modell B1.

Die vier Tasten erfüllen folgende Funktionen:

a) START
Das Niederdrücken dieser Taste bewirkt, daß Karten vom Magazin zugeführt werden und, wenn alle Elemente des Kartengerätes in Ordnung sind, daß die Lampe BETRIEB aufleuchtet. Nach dem Drücken der Tasten STOP oder AUSLAUF dient die Taste START zur Wiederherstellung des betriebsbereiten Zustandes.

b) STOP
Das Drücken dieser Taste hat zur Folge, daß die Maschine nach Beendigung der gerade laufenden Operation anhält; außerdem verlöscht die Anzeigelampe BETRIEB.

c) AUSLAUF [1])
Nach Entleeren des Kartenmagazins bewirkt das Drücken dieser Taste, daß die zwei in der Kartenbahn befindlichen Karten in das Ablagefach 1 auslaufen, ohne daß sie gestanzt oder gelesen werden. Um einen aufgetretenen Kartenanstoß zu berichtigen, sind demnach folgende Handgriffe vorzunehmen: Zunächst sind die Karten aus dem Kartenmagazin zu entfernen und die Taste AUSLAUF ist zu drücken. Die beiden ohne Verarbeitung abgelegten Karten sind anschließend wieder ins Kartenmagazin zu packen; dahinter sind dann die aus dem Magazin entnommenen Karten zu plazieren. Zum Schluß ist erneut die Taste START zu drücken. Da das Drücken von AUSLAUF auch die Anzeigelampe LETZTE KARTE zum Verlöschen bringt, sollte im Bedarfsfall auch diese Taste noch einmal gedrückt werden.

d) LETZTE KARTE
Wenn das Kartenmagazin leer ist, ermöglicht das Drücken dieser Taste das Lesen und Stanzen der letzten Karten und ihre Ablage. Das Drücken dieser Taste bewirkt außerdem, daß die Bitstelle des Statusbytes, die die Geräteausnahmebedingung anzeigt, in dem Augenblick auf 1 gesetzt wird, in dem der erste Befehl, gleichgültig, ob Lesen oder Schreiben, zur Ausführung gelangen soll, obwohl die letzte Karte bereits gelesen wurde. Gilt nur für das Modell B1.

6.4. Das Kartengerät IBM 2540

Im Gegensatz zu den vorher beschriebenen Kartengeräten wird die IBM 2540, die in Abb. 6.7. vorgestellt wird, über eine gesonderte Steuereinheit, nämlich über die IBM 2821 (Modelle 1, 4 oder 5) an die IBM /370 angeschlossen. Da Kartengerät und Steuereinheit logisch als eine einzige Einheit angesehen werden können, werden wir nicht zwischen den Funktionen, die von der IBM 2821 ausgeübt werden, und den Funktionen, die der IBM 2540 zugeordnet sind, eine Unterscheidung vornehmen.

1) Non Process - Run Out (Auslauf ohne Verarbeitung)

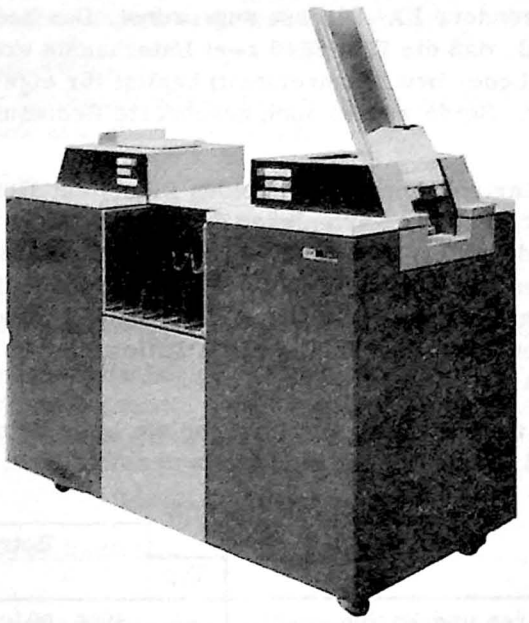
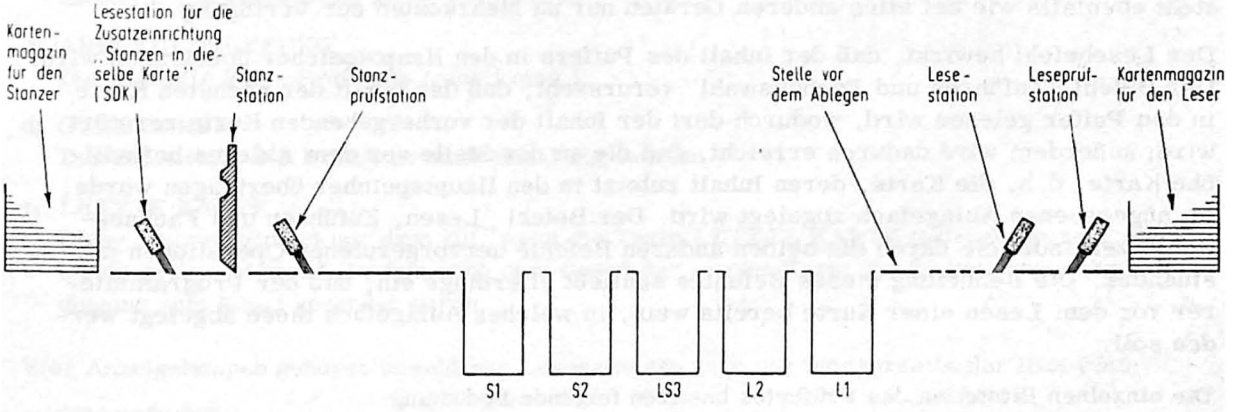


Abb. 6.7. Das Kartengerät IBM 2540

Von der logischen Seite her gesehen, bildet die IBM 2540 zwei Einheiten. Rechts befindet sich die Abfühleinheit, die die Karten zeilenweise mit einer Geschwindigkeit von 1000 cd/m liest. Die Karten passieren dabei zwei Sätze Abfühlbürsten und wandern anschließend in eines von den drei rechten Ablagefächern. Wegen dieser hohen Geschwindigkeit fassen die Radialablagefächer 3000 Karten. Bei der links befindlichen Stanzeinheit laufen die Karten vom Magazin unter zwei Sätzen von Abfühlbürsten und der Stanzstation hindurch bis in eines von den drei linken Ablagefächern. Das mittlere Ablagefach kann also von den beiden Zuführungen gemeinsam benutzt werden, die ansonsten höchst unabhängig voneinander arbeiten. Die Lesestation auf der Stanzerseite ist eine Sonderausrüstung, die Mehrkosten bedingt. Diese Sonderausrüstung ist unter der Bezeichnung "Stanzen in gleiche Karte" bekannt.



Anmerkung: L Ablagefach für den Leser (Abfühler)
S Ablagefach für den Stanzer

Abb. 6.8. Die Kartenbahnen bei der IBM 2540

Jeder Einheit ist eine besondere EA-Adresse zugeordnet. Das bedeutet beim Anschluß an den Bytemultiplexkanal, daß die IBM 2540 zwei Unterkanäle erfordert. Jede der beiden logischen Einheiten (Lese- bzw. Stanzeinheit) besitzt ihr eigenes Statusbyte und auch ihr eigenes Prüfbyte. Beide weisen auch gesonderte Bedienungs- und Kontrollorgane auf.

Die Abb. 6.8. zeigt die Kartenbahnen in den beiden Einheiten. Beide Kartenzuführungen sind voll gepuffert. Somit liest der Kartenleser der IBM 2540 den Karteninhalt in seinen Puffer unabhängig von anderen Einheiten ein. Wird nun von der IBM /370 ein Befehl zum Lesen einer Karte gegeben, so werden die Informationen vom Puffer in den Hauptspeicher übertragen. Sobald der Puffer geleert ist, wird eine andere Karte gelesen. Der Kartenstanzer benutzt, wie schon gesagt, ebenfalls einen Puffer, der in ähnlicher Weise wie bei der IBM 2520 arbeitet.

Wir wollen zuerst den Kartenleser und anschließend die Lese-Stanzeinrichtung betrachten. Die Befehlsschlüssel für den Leser sind die in der Abb. 6.9. aufgeführten.

		Betriebsart	
	Typ	1	2
Lesen, Zuführen und Fachauswahl	AA	SS00 0010	SS10 0010
Lesen	AB	1100 0010	1110 0010
Zuführen und Fachauswahl	BA	SS10 0011	——
Prüfen	——	0000 0100	——
Keine Operation (Nop)	——	0000 0011	——

Abb. 6.9. Die Befehlsschlüssel der Leseeinheit der IBM 2540

Wird für SS die Ziffernkombination 00 gesetzt, so wird das Ablagefach L1 angesprochen, SS gleich 01 spezifiziert das Ablagefach L2 und SS gleich 10 das Ablagefach LS3. Die einzelnen Befehle müssen so gestaffelt werden, daß einem Befehl des Typs AA ein solcher des Typs AA oder des Typs AB folgt. Einem Befehl des Typs AB muß ein anderer Befehl des gleichen Typs oder ein Befehl des Typs BA folgen. Schließlich muß sich einem Befehl des Typs BA ein Befehl des Typs AA oder des Typs AB anschließen. Die Betriebsart 2 steht ebenfalls wie bei allen anderen Geräten nur zu Mehrkosten zur Verfügung.

Der Lesebefehl bewirkt, daß der Inhalt des Puffers in den Hauptspeicher übertragen wird. Der Befehl „Zuführen und Fachauswahl" verursacht, daß der Inhalt der nächsten Karte in den Puffer gelesen wird, wodurch dort der Inhalt der vorhergehenden Karte zerstört wird; außerdem wird dadurch erreicht, daß die an der Stelle vor dem Ablegen befindliche Karte, d.h. die Karte, deren Inhalt zuletzt in den Hauptspeicher übertragen wurde, im angegebenen Ablagefach abgelegt wird. Der Befehl „Lesen, Zuführen und Fachauswahl" verbindet die durch die beiden anderen Befehle hervorgerufenen Operationen miteinander. Die Benutzung dieses Befehles schließt allerdings ein, daß der Programmierer vor dem Lesen einer Karte bereits weiß, in welches Ablagefach diese abgelegt werden soll.

Die einzelnen Bitstellen des Prüfbytes besitzen folgende Bedeutung:

a) Bitstelle 0 —— Befehlszurückweisung
Diese Bitstelle wird auf 1 gesetzt, wenn entweder eine ungültige Befehlsfolge oder ein nicht definierter Befehlsschlüssel vorliegt.

b) Bitstelle 1 —— Eingriff erforderlich

c) Bitstelle 2 — Fehler in der Ausgangsleitung

d) Bitstelle 3 — Ausrüstungsfehler
Diese Bitstelle wird nur zur Kanalendezeit auf 1 gesetzt; die Bedingung betrifft also diejenige Karte, deren Daten gerade erst übertragen wurden.

e) Bitstelle 4 — Datenfehler
Wird bei der Betriebsart 1 eine Kartenspalte mit mehr als einer Lochung in den Zeilen 1 bis 7 entdeckt, so wird dieser Bitstelle eine 1 zugewiesen.

f) Bitstelle 5
Diese Bitstelle wird bei der IBM 2540 nicht verwendet, da ein Datenverlust nicht auftreten kann.

g) Bitstelle 6 — ungewöhnliche Befehlsfolge
Diese Bitstelle wird dann auf 1 gesetzt, wenn ein Befehl des Typs AB einem anderen Befehl des gleichen Typs folgt, ohne daß zwischen diesen beiden Befehlen ein Befehl des Typs BA eingeschoben ist.

h) Bitstelle 7
Diese Bitstelle besitzt keine Bedeutung.

Die Bedienungs- und Kontrollorgane der Leseseite der IBM 2540 sind in Abb. 6.10. dargestellt. Fünf Anzeigelampen sind allein zur Anzeige des Zustandes der Leseseite installiert:

a) MAGAZIN
Diese Lampe leuchtet auf, wenn es zu einem Kartenanstoß, einer fehlerhaften Kartenzuführung, einem Unterbleiben der Kartenzuführung usw. gekommen ist; gleichzeitig wird der Bitstelle 1 des Prüfbytes eine 1 zugewiesen.

b) BETRIEB
Durch ihr Aufleuchten wird angezeigt, daß die Karteninhalte von den Lesestationen aufgenommen werden (die erste Lesestation ist eine Prüfstation, die die Richtigkeit des Lesens überprüft; tatsächlich wird der Karteninhalt erst von der zweiten Lesestation in den Puffer eingelesen). Ferner kann beim Aufleuchten dieser Lampe das Magazin nicht leer sein, es sei denn, daß die Lampe LETZTE KARTE gleichzeitig ebenfalls aufleuchtet. Schließlich darf auch die Bitstelle 1 des Prüfbytes keine 1 enthalten.

c) ABFUEHLPRUEFUNG
Die Bitstelle 3 des Prüfbytes enthält eine 1.

d) GUELTIGKEIT
Der Bitstelle 4 des Prüfbytes wird eine 1 zugewiesen.

e) LETZTE KARTE
Diese Lampe leuchtet nur dann auf, wenn die Taste LETZTE KARTE gedrückt wurde; sie verlöscht in dem Moment, in dem nach dem Lesen der letzten Karte die Einheitenausnahme-Bedingung zum Kanal gesendet wurde.

Vier Anzeigelampen gehören sowohl zur Leseseite als auch zur Stanzerseite der IBM 2540:

a) TRANSPORT
Es liegt ein Kartenanstoß im Bereich vor den Ablagefächern vor. Die Bitstelle 1 des Prüfbytes wird auf 1 gesetzt.

b) NETZ
Das Aufleuchten dieser Lampe zeigt an, daß die IBM 2540 mit Spannung versorgt ist.

c) KARTENABLAGE
Das Aufleuchten dieser Lampe zeigt an, daß ein Ablagefach gefüllt ist. Die Bitstelle 1 des Prüfbytes außerdem auf 1 gesetzt.

d) SICHERUNG
Durch das Aufleuchten dieser Lampe wird angezeigt, daß eine Sicherung durchgebrannt ist. Zum Einsetzen einer neuen sollte das Wartungspersonal gerufen werden, damit bei dieser Gelegenheit das Gerät überprüft werden kann. Außerdem wird die Bitstelle 1 des Prüfbytes auf 1 gesetzt.

Es werden nur drei Tasten benutzt:

a) START
Das Drücken der Taste START zu Arbeitsbeginn bewirkt, daß Karten eingezogen werden, es versetzt außerdem den Leser in den Zustand BETRIEB. Ist der Leser durch Drücken der Taste STOP angehalten worden, so stellt das erneute Drücken der Taste START den Zustand BETRIEB wieder her. Wenn die Lampe LETZTE KARTE nicht aufleuchtet, das Magazin leer und die Magazinklappe geöffnet ist, so wird durch das Drücken dieser Taste ein Auslauf der in der Kartenbahn befindlichen Karten erreicht, ohne daß diese noch verarbeitet werden.

b) STOP
Durch das Drücken dieser Taste wird die Maschinenoperation abgestoppt und die Lampe LETZTE KARTE ausgeschaltet.

c) LETZTE KARTE
Das Drücken dieser Taste läßt die Lampe LETZTE KARTE aufleuchten, wodurch das Lesen der letzten Karten ermöglicht wird.

Abb. 6.10. Die Bedienungs- und Kontrollorgane der Leseseite der IBM 2540

Für die Stanzeinheit der IBM 2540 gelten die Befehlsschlüssel, die in der Abb. 6.11. aufgeführt sind.

	Typ	Betriebsart 1	Betriebsart 2
Lesen (Abfühlen)	AB	1100 0010	
Lesen an der Station SDK, Stanzen, Zuführen und Fachauswahl	BA	SS00 1001	SS10 1001
Stanzen, Zuführen und Fachauswahl	BB	SS00 0001	SS10 0001
Prüfen	—	0000 0100	
Keine Operation (Nop)	—	0000 0011	

Anm.: SDK → Stanzen in gleiche Karte
(Stanzen in dieselbe Karte)

Abb. 6.11. Die Befehlsschlüssel der Stanzeinheit der IBM 2540

Es können nur die folgenden Befehlsfolgen gebildet werden:

a) Einem Befehl des Typs BB muß ein Befehl des Typs BB oder des Typs BA folgen.

b) Einem Befehl des Typs BA kann nur ein Befehl des Typs AB folgen.

c) Einem Befehl des Typs AB kann ein Befehl des Typs AB oder ein Befehl des Typs BA oder ein Befehl des Typs BB folgen.

Die Befehle „Prüfen" und „Nop" bleiben bei der Bestimmung einer richtigen Befehlsfolge unberücksichtigt. Setzt man für SS die Ziffern 00 ein, so ist damit das Ablagefach S1 gemeint; durch SS gleich 01 ist S2 festgelegt und durch SS gleich 10 das Ablagefach LS3.

Der Befehl „Stanzen, Zuführen und Fachauswahl" bewirkt, daß Daten vom Hauptspeicher zum Puffer gesendet werden. Anschließend wird ein Kartentransport eingeleitet, dabei wird der Pufferinhalt in die Karte gestanzt, die sich vor der Stanzstation befand. Die Karte, die vor der Station SDK lag, wird bei dieser Operation nicht in den Puffer gelesen. Die

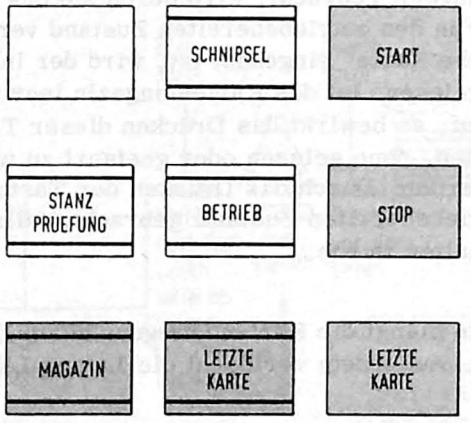


Abb. 6.12. Die Bedienungs- und Kontrollorgane der Stanzeinheit der IBM 2540

Fachauswahl bezieht sich auf die zu stanzende Karte. Ereignet sich jedoch beim Stanzen dieser Karte ein Fehler, so wird sie automatisch ins Ablagefach S1 abgelegt.

Der Befehl „Lesen an der Station SDK, Stanzen, Zuführen und Fachauswahl" erfüllt die gleichen Funktionen wie der zuvor beschriebene Befehl; zusätzlich kommt hinzu, daß der Inhalt der vor der Station SDK liegenden Karte zur gleichen Zeit in den dazugehörigen Puffer gelesen wird, während die vor der Stanzstation liegende Karte gestanzt wird.

Das Gerätestatusbyte und das Prüfbyte unterscheiden sich nicht von den gleichen Bytes der Leseseite. Man beachte jedoch, daß das Bit 6 des Prüfbytes dann und nur dann gleich 1 ist, wenn zwei Leseoperationen ausgeführt werden sollen, ohne daß zwischen sie ein anderer Befehl geschoben ist. Wenn die Einrichtung „Stanzen in gleiche Karte" nicht eingebaut ist, so haben die Befehle „Lesen" und „Lesen an der Station SDK, Stanzen, Zuführen und Fachauswahl" keine Gültigkeit.

Die Bedienungs- und Kontrollorgane für die Stanzeinheit der IBM 2540 sind in der Abb. 6.12. dargestellt.

Die fünf Anzeigelampen haben folgende Bedeutung:

a) SCHNIPSEL
Der Schnipselkasten ist voll. Die Prüfbitstelle 1 enthält eine 1.

b) BETRIEB
Die Bitstelle 1 des Prüfbytes ist nicht auf 1 gesetzt.

c) STANZPRUEFUNG
Die Bitstelle 3 des Prüfbytes wird auf 1 gesetzt.

d) MAGAZIN
Bei Aufleuchten dieser Lampe liegt ein Kartenanstoß oder ein Versagen der Kupplung vor. Die Bitstelle 1 des Prüfbytes enthält eine 1.

e) LETZTE KARTE
Diese Lampe hat nur dann die übliche Bedeutung, wenn die Zusatzeinrichtung "Stanzen in die gleiche Karte" installiert ist.

Die drei Tasten erfüllen folgende Funktionen:

a) START
Zu Beginn der Maschinenarbeit gedrückt, wird durch sie das Einziehen der Karten eingeleitet und die Maschine in den betriebsbereiten Zustand versetzt. Wenn die Einrichtung „Stanzen in die gleiche Karte" eingebaut ist, wird der Inhalt der ersten eingezogenen Karte in den Puffer gelesen. Ist das Kartenmagazin leer und leuchtet die Lampe LETZTE KARTE nicht auf, so bewirkt das Drücken dieser Taste, daß alle in der Kartenbahn befindlichen Karten, ohne gelesen oder gestanzt zu werden, in das Ablagefach S1 transportiert werden. Durch das Drücken der Taste START wird die Maschine auch wieder in den betriebsbereiten Zustand gebracht, falls sie zuvor durch das Drücken von STOP angehalten wurde.

b) STOP
Das Drücken dieser Taste bringt die Kartenbewegungen am Ende der gerade laufenden Operation zum Stillstand. Außerdem verlöscht die Lampe LETZTE KARTE.

c) LETZTE KARTE
Diese Taste besitzen nur diejenigen Kartengeräte, in die die Zusatzeinrichtung „Stanzen in gleiche Karte" eingebaut ist. Durch Niederdrücken dieser Taste wird die Zuführung der letzten Karte ermöglicht; die Anzeigelampe LETZTE KARTE leuchtet auf.

6.5. Die Kartenleser IBM 3504 und IBM 3505

Die Abb. 6.13. zeigt eine schematische Darstellung des Kartenlesers IBM 3505. Die Kartenbahn selbst wird in der nachfolgenden Abb. 6.14. gezeigt. Der Kartenleser IBM 3504 ist in Aussehen und Wirkungsweise mit dem Kartenleser IBM 3505 identisch. Die IBM 3504 wird nicht an einen Kanal angeschlossen. Sie wird stattdessen an eine integrierte Kanalsteuereinheit angeschlossen, die für die kleineren Modelle der Computerfamilie IBM /370 verfügbar ist. Ein solcher integrierter Anschluß reduziert die Systemkosten. Die IBM 3505, Modell B1 und die IBM 3504, Modell A1, lesen die Karten mit einer Geschwindigkeit von 800 cd/m. Bei den Modellen B2 bzw. A2 ist diese auf 1200 cd/m gesteigert.

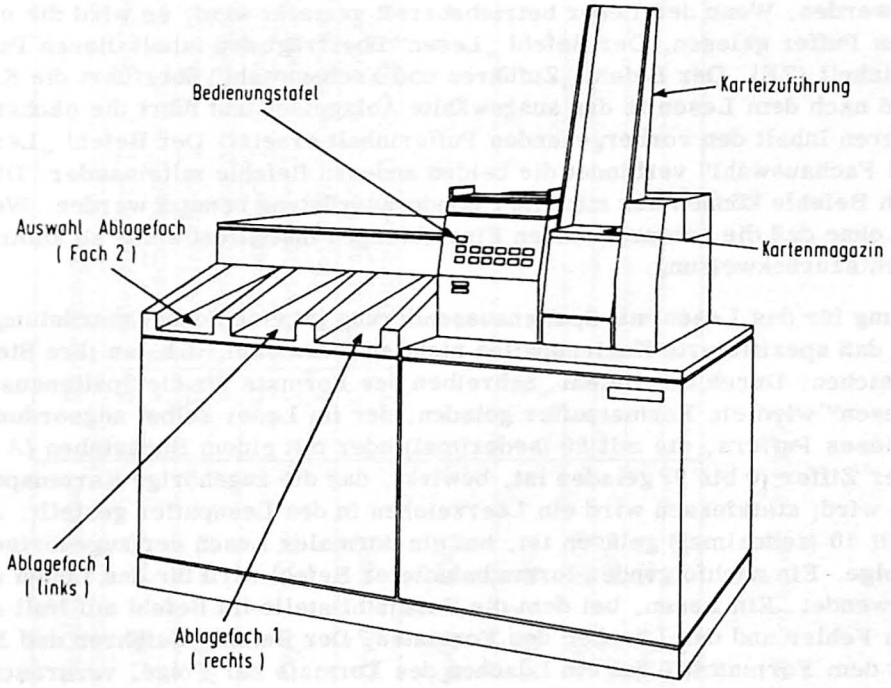


Abb. 6.13. Schematische Darstellung der IBM 3505

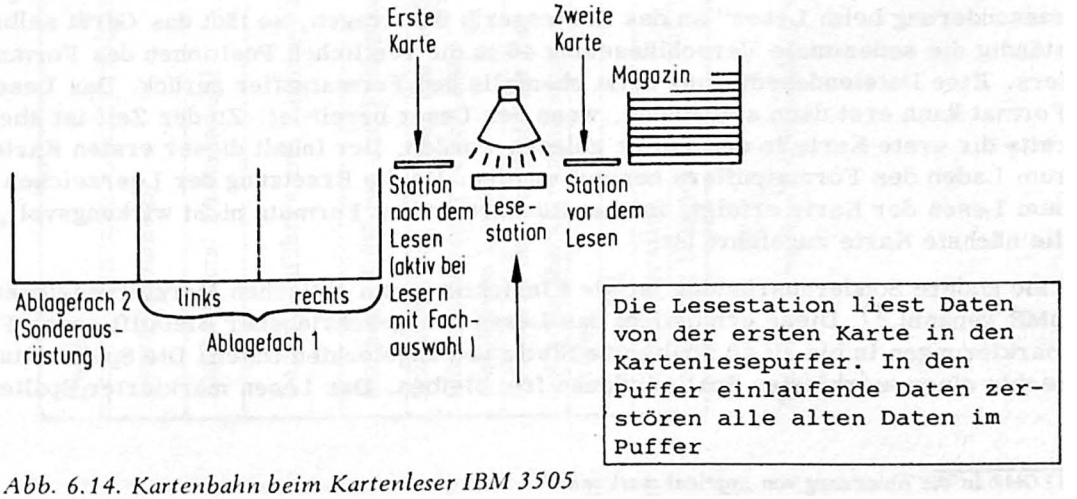


Abb. 6.14. Kartenbahn beim Kartenleser IBM 3505

Das Ablagefach 1 setzt sich physisch aus zwei Ablagefächern zusammen. Mittels eines Schalters auf der Schalterplatte wird dasjenige Fach eingestellt, das benutzt werden soll. Wenn das ausgewählte Fach mit 1750 Karten gefüllt ist, schaltet der Kartenleser automatisch auf das andere Fach um. Bei Einschaltung des Kartenlesers wird das rechte Ablagefach zuerst benutzt.

Das Ablagefach 2 stellt eine Sonderausrüstung dar, die nur für zusätzliche Kosten installiert wird. Fehlt das Ablagefach 2, so existiert die Station nach der Leseeinrichtung, wie sie in Abb. 6.14. gezeigt ist nicht, und alle Karten laufen unmittelbar in das Ablagefach 1, welches Ablagefach auch immer im Kanalbefehlsschlüssel aufgeführt ist.

Die Befehlsschlüssel für die Kanalbefehle sind in der Abb. 6.15. zusammengestellt. Beim Ansprechen des Ablagefaches 2 kann stets einer von zwei verschiedenen Schlüsseln benutzt werden. Wenn der Leser betriebsbereit gemacht wird, so wird die erste Karte in einen Puffer gelesen. Der Befehl „Lesen" überträgt den Inhalt dieses Puffers zur Zentraleinheit (ZE). Der Befehl „Zuführen und Fachauswahl" überführt die Karte an der Station nach dem Lesen in das ausgewählte Ablagefach und führt die nächste Karte zu, so daß deren Inhalt den vorhergehenden Pufferinhalt ersetzt. Der Befehl „Lesen, Zuführen und Fachauswahl" verbindet die beiden anderen Befehle miteinander. Die formatbehafteten Befehle können nur mit einer Sonderausrüstung benutzt werden. Werden sie gegeben, ohne daß die entsprechenden Einrichtungen installiert sind, so kommt es zu einer Befehlszurückweisung.

Die Einrichtung für das Lesen mit Spaltenaussonderung ist eine Sonderausrüstung. Sie gestattet es, daß spezifizierte Kartenspalten nicht zu lesen sind, d.h. an ihre Stelle treten Leerzeichen. Durch den Befehl „Schreiben des Formats für die Spaltenaussonderung beim Lesen" wird ein Formatpuffer geladen, der im Leser selbst angeordnet ist. Jede Stelle dieses Puffers, die mit 80 (sedezimal) oder mit einem Buchstaben (A bis Z) oder mit einer Ziffer (0 bis 9) geladen ist, bewirkt, daß die zugehörige Kartenspalte nicht gelesen wird; stattdessen wird ein Leerzeichen in den Lesepuffer gestellt. Jede Stelle, die mit 40 (sedezimal) geladen ist, hat ein normales Lesen der zugehörigen Kartenspalte zur Folge. Ein nachfolgender formatbehafteter Befehl wird für das Lesen und Zuführen verwendet. Ein Lesen, bei dem die Formatbitstelle im Befehl auf Null steht, bewirkt einen Fehler und das Löschen des Formates. Der Befehl „Zuführen und Fachauswahl" mit dem Formatbit 0 hat ein Löschen des Formats zur Folge, verursacht jedoch keinen Fehler. Ein formatbehafteter Befehl (die Formatbitstelle enthält eine 1), der gegeben wird, ohne daß ein Format geladen ist, führt zu einer Befehlszurückweisung. Werden weniger als 80 Bytes durch den Befehl „Schreiben des Formats für die Spaltenaussonderung beim Lesen" an das Kartengerät übertragen, so lädt das Gerät selbstständig die sedezimale Verschlüsselung 40 in die restlichen Positionen des Formatpuffers. Eine Dateiendebedingung setzt ebenfalls den Formatpuffer zurück. Das Lesen nach Format kann erst dann stattfinden, wenn der Leser bereit ist. Zu der Zeit ist aber bereits die erste Karte in den Puffer gelesen worden. Der Inhalt dieser ersten Karte kann zum Laden des Formatpuffers benutzt werden. Da die Ersetzung der Leerzeichen nach dem Lesen der Karte erfolgt, ist das Rücksetzen des Formats nicht wirkungsvoll, bis die nächste Karte zugeführt ist.

Eine andere Sonderausrüstung ist die Einrichtung zum optischen Markierungslesen, OMR genannt.[1] Diese ermöglicht das Lesen handgeschriebener Bleistift- oder Federmarkierungen in bis zu 40 Spalten an Stelle von abgelochten Daten. Die Spalten links und rechts einer markierten Spalte müssen frei bleiben. Das Lesen markierter Spalten er-

1) OMR ist die Abkürzung von „optical mark read"

Befehl	Befehlsschlüssel		Bemerkungen	
Lesen (Abfühlen)	11x0	y010	x y	0→Betriebsart 1, 1→Betriebsart 2 0→Lesen ohne Format, 1 Lesen mit Format
Zuführen und Fachauswahl	xx10	y011	xx y	00→Ablagefach 1, 01 und 10 Ablagefach 2, 11 ungültig 0→Pufferladen ohne Format, 1→Pufferladen mit Format
Lesen, Zuführen und Fachauswahl	xxy0	z010	xx y z	00→Ablagefach 1, 01 und 10→Ablagefach 2, 11 ungültig 0→Betriebsart 1, 1→Betriebsart 2 0→Pufferladen ohne Format, 1→Pufferladen mit Format
Stanzen, Zuführen und Fachauswahl	xxy0	0001	xx y	00→Ablagefach 1, 01 und 10→Ablagefach 2, 11 ungültig 0→Betriebsart 1, 1→Betriebsart 2
Schreiben des Formats für die Spaltenaussonderung beim Lesen (selektives Lesen)	0001	0001		
Schreiben des Formats für das optische Markierungslesen	0011	0001		
Prüfen	0000	0100		
Testen EA	0000	0000		
Keine Operation	0000	0011		
Lesen zur Diagnostik	1101	0010		

Abb. 6.15. Die Befehlsschlüssel bei der IBM 3504 und bei der IBM 3505

folgt unter Steuerung eines Formats, das durch den Befehl „Schreiben des Formats für das optische Lesen von Markierungen" geladen wird. Wenn durch diesen Befehl der Puffer mit anderen Zeichen geladen ist als mit den sedezimal 40 bzw. 80 verschlüsselten, so muß der nächste Befehl der Befehl „Zuführen und Fachauswahl" sein, anderenfalls kommt es zum Auftreten eines Ausrüstungsfehlers. Die OMR- und die RCE-Einrichtung [1] können nicht zur gleichen Zeit benutzt werden. Pufferwerte von sedezimal 80 oder von A bis Z bzw. 0 bis 9 verursachen das optische Lesen von Markierungen in dieser Spalte. Wird eine Spalte als Markierungsspalte deklariert, so wird der Inhalt der rechts von ihr gelegenen Spalte nicht zur ZE übertragen. Somit kann durch eine volle Karte nicht die Gesamtzahl von 80 Spalten eingelesen werden. Freilich werden zwar 80 Bytes übertragen, doch jeweils Leerzeichen von den rechts von Markierungen gelegenen Spalten.

Wenn der Leser eine zweifelhafte Markierung in einer Spalte entdeckt, so sendet er die sedezimale Verschlüsselung 3F (in der Betriebsart 2 jedoch 3F3F) für die betreffende Spalte und setzt außerdem das gleiche Zeichen auf Byte 80 (bzw. auf Byte 160 bei der Betriebsart 2). Für zweifelhafte Spalten wird kein Maschinenfehler angezeigt.

Das Gerätestatusbyte gleicht dem der IBM 2501 (siehe Abschnitt 6.2.).

Die einzelnen Bitstellen des Prüfbytes 0 haben folgende Bedeutung:

a) Bitstelle 0 —— Befehlszurückweisung
Eine 1 wird dieser Bitstelle bei ungültigem Befehlsschlüssel, bei einem Befehlsschlüssel einer nicht installierten Sonderausrüstung und bei einem Befehl zugewiesen, der nicht in der richtigen Reihenfolge erscheint.

b) Bitstelle 1 —— Eingriff erforderlich
Fehler in der Kartenzuführung, leeres Kartenmagazin, gefülltes Ablagefach, geöffnete Maschinenabdeckung oder gedrückte Stoptaste sind die Ursachen der Zuweisung einer 1 zu dieser Bitstelle.

c) Bitstelle 2 —— Fehler in der Ausgangsleitung

d) Bitstelle 3 —— Ausrüstungsfehler

e) Bitstelle 4 —— Datenfehler
Bei mehr als einer Lochung in den Zeilen 1 bis 7 einer Spalte kommt es in der Betriebsart 1 zur Zuweisung einer 1 zu dieser Bitstelle.

f) Bitstelle 5 —— nicht benutzt

g) Bitstelle 6 —— Anomale Formatrücksetzung
Eine 1 wird dieser Bitstelle zugewiesen, wenn beim Formatmodus ein unformatiertes Lesen oder Lesen, Zuführen und Fachauswahl erfolgt. Der betreffende Befehl wird ausgeführt und danach das Format zurückgesetzt.

h) Bitstelle 7 —— Dauerfehler
Der Maschinenbediener hat die Dauerfehlertaste gedrückt.

Die Prüfbytes 1, 2 und 3 werden von der Maschine bei der automatischen Fehlerbeseitigung beansprucht. Die IBM 3504 und 3505 versuchen bestimmte Fehler selbst zu beseitigen, indem sie z.B. bis zu drei Kartenzuführungszyklen durchführen, bevor dem Kanal ein Fehler gemeldet wird.

1) RCE ist die Abkürzung von „read column eliminate" (Lesen mit Spaltenaussonderung)

In der Abb. 6.16. sind die Bedienungs- und Kontrollorgane für das Maschinenbedienungspersonal dargestellt. Wir wollen sie kurz beschreiben:

a) KARTENABLAGE 1 BEREIT (L R)
Legt man diesen Hebelschalter auf L, so erfolgt der Übergang zum linken Ablagefach, wenn das rechte voll ist, wenn auf R gelegt, so erfolgt der Übergang aufs linke Ablagefach.

Das Leuchtfeld „LINKS AKTIV" oder „RECHTS AKTIV" zeigt an, welches Ablagefach gerade benutzt wird.

b) START, STOP, LETZTE KARTE und KARTENAUSLAUF
Diese Tasten funktionieren in gleicher Weise wie bei der IBM 2501 und bei der IBM 2520.

c) LAMPENPRÜFUNG
Das Drücken dieser Taste bewirkt das Aufleuchten aller Lampen. Sie wird benutzt, um ausgebrannte Lampen zu erkennen und auswechseln zu können.

d) ABDKG OFFEN
Hiermit kann die Haubenabdeckung aufgeklinkt werden.

e) DAUERFEHLER
Hiermit wird die Bitstelle 7 des Prüfbytes auf 1 gesetzt. Ein erneutes Drücken setzt diese Bitstelle zurück auf 0. Die Bitstelle 0 des Prüfbytes 1 bleibt jedoch ungeändert auf 1; sie wird nur ausgeschaltet bei Ausführung eines Befehls, der nicht den Befehlen „Prüfen" oder „keine Operation" gleicht.

f) LOG OUT
Das Drücken dieser Taste zeigt den Inhalt des Prüfbytes 2 an den Leuchtfeldern 8, 4, 2, 1 an.

Die übrigen Anzeigelampen haben folgende Bedeutung:

a) NETZ EIN
Der Leser ist mit Spannung versorgt.

b) LETZTE KARTE
Drücken der Taste LETZTE KARTE.

c) BEREIT
Betriebsbereitschaft des Lesers.

d) EINGRIFF ERFORD
Es liegt ein Problem vor, das den Eingriff der Maschinenbedienung erfordert. Näheres ist aus den Schriftzeichen zu ersehen, die auf der Tafel von hinten beleuchtet sind. Sie verweisen auf folgende Probleme:

— TEMP
Überhitzte Leselampe
— KARTENABLAGE
Das Ablagefach ist gefüllt oder der Hebelschalter vom Fach 1 zeigt nicht auf ein leeres Ablagefach
— ABDKG OFFEN
Die Abdeckhaube ist nicht geschlossen
— KARTENMAGAZIN
Das Magazin ist leer: entweder Karten nachlegen oder Taste LETZTE KARTE drücken
— KARTE PRÜFEN
Gleicht dem Kartenauslauf: die erste von zwei Karten ist die ungeeignete, es sei

denn, WECHSEL 1 leuchtet ebenfalls auf; in diesem Fall ist die letzte Karte die fehlerhafte. Es ist auf Verlochung (nicht mehr als eine Lochung in den Zeilen 1 bis 7), verzogene Lochungen usw. zu prüfen

— TRANSPORT
Die Karten sind durcheinander geraten oder schlecht zugeführt. AUSLAUF drücken und beide Karten zurück ins Kartenmagazin legen

— FORMAT LÖSCHEN
Im Formatmodus sind Lesen bzw. Lesen, Zuführen und Fachauswahl ohne gesetzte Formatbitstelle veranlaßt worden

— WECHSEL 1
Die letzte ausgelaufene Karte ist die Karte, die das angezeigte Ärgernis verursacht hat

— KARTENAUSLAUF
Beide Karten zurück ins Magazin legen und START drücken. Damit ist die Maschine vorbereitet, erneut die Arbeit zu versuchen

— KARTENANSTOSS
Kartenbruch u. a. m.

— MASCH PRFG
Das Drücken der Taste LOG OUT zeigt den Inhalt des Prüfbytes 2 auf den Leuchtfeldern 8, 4, 2, 1 an. Leuchtet KARTENAUSLAUF ebenfalls auf, so empfiehlt sich eine Wiederholung, da die Fehlerbeseitigung möglich sein kann

— DAUERFEHLER
Leuchten TRANSPORT und KARTENANSTOSS ebenfalls auf, so ist LOG OUT zu drücken und die Daten aufzuzeichnen. Die Bedingung ist zu untersuchen

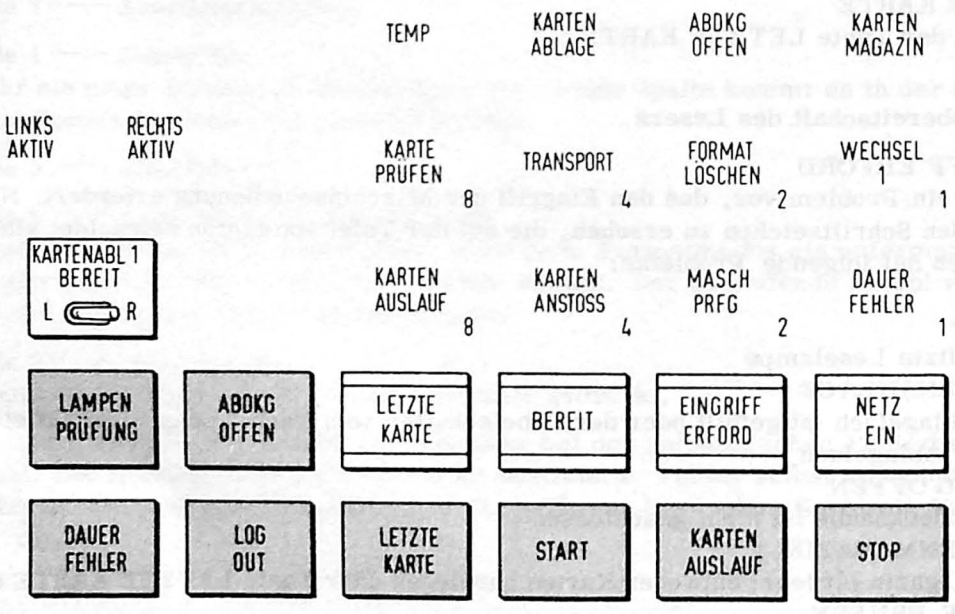


Abb. 6.16. Die Bedienungs- und Kontrollorgane der IBM 3504 und IBM 3505

6.6. Der Kartenstanzer IBM 3525

Der Kartenstanzer IBM 3525 kann an eine IBM 3505 oder an eine IBM 3504 angeschlossen werden. Er wird in drei Modellen ausgeliefert. Das Modell P1 stanzt 100 Karten pro Minute, Modell 2 stanzt 200 Karten pro Minute und das Modell 3 bringt es auf 300 Karten pro Minute. Die Kartenbahn ist in der Abb. 6.17. dargestellt. Die Lesestation ist eine Sonderausrüstung. Ist sie installiert, so ist das Lesen mit Spaltenaussonderung eine Zusatzeinrichtung. Das Lesen von Markierungen auf optischer Basis ist nicht verfügbar. Eine Druckeinrichtung zum Drucken von bis zu 64stelligen Zeilen kann geordert werden. Es gibt zwei verschiedene:

a) Drucken auf den Druckzeilen 1 und 3
b) Drucken auf allen 25 Druckzeilen auf einer Karte.

Die Druckzeilen sind dabei wie folgt angeordnet:

Druckzeile	Anordnung
1	Über der 12er Zeile
2	Auf der 12er Zeile
3	Zwischen 12er und 11er Zeile
4	Auf der 11er Zeile
5	Zwischen der 11er und 0er Zeile
.	
.	
.	
24	Auf der 9er Zeile
25	Unterhalb der 9er Zeile

Wird nur eine Zeile gedruckt, so wird die Geschwindigkeit der IBM 3525 nicht betroffen. Beim Drucken von zwei Zeilen sind die Kartengeschwindigkeiten bei den einzelnen Modellen 100, 200 bzw. 240 Karten pro Minute. Beim Bedrucken von allen 25 Zeilen sinken sie auf 24, 29 und 30 Karten pro Minute herab.

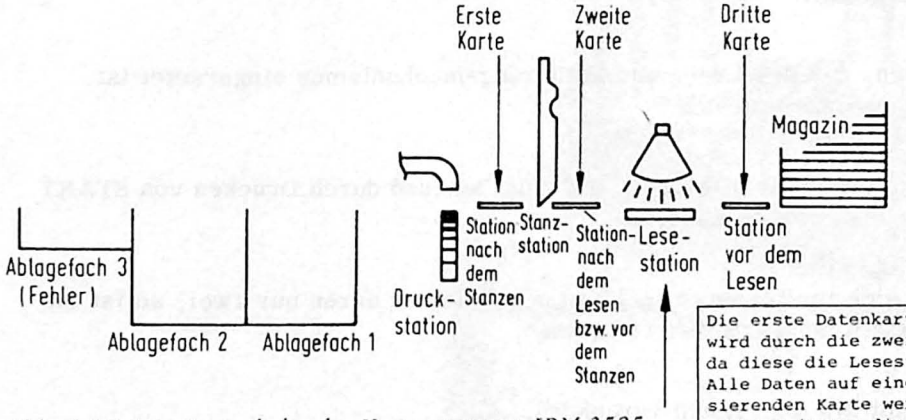


Abb. 6.17. Die Kartenbahn des Kartenstanzers IBM 3525

Zu den Befehlen ist folgendes zu sagen, wobei wir sie mit denen der IBM 3505 vergleichen wollen. Die Befehle „Zuführen und Fachauswahl" sowie „Keine Operation" entsprechen einander. Ist die Leseeinrichtung bzw. die Leseeinrichtung mit selektivem Lesen installiert, so gleichen sich auch die hierfür geltenden Befehle. Der Befehl

„Prüfen" greift zu acht Bytes zu. Das erste Prüfbyte hat den gleichen Aufbau wie das der IBM 3505; die anderen werden zur Festhaltung und Analyse von Maschinenfehlern benötigt. Da optisches Markierungslesen nicht möglich ist, ist der entsprechende Befehl der einzige, der von der IBM 3505 nicht übernommen ist. Das Stanzen wird durch den Befehl „Stanzen, Zuführen und Fachauswahl" veranlaßt. Sein Befehlsschlüssel ist xxy00001. Die Bitstellen xx zeigen die Fachauswahl an: 00 führt zur Ablage im Fach 1, 01 oder 10 zur Ablage im Fach 2. Die Betriebsart wird durch y angezeigt (0 —► Betriebsart 1, 1 —► Betriebsart 2). Bei der Installation der Druckeinrichtung gibt es noch den Befehl „Drucken Zeile" mit dem Befehlsschlüssel zzzzz101. Unter zzzzz ist die Zeilennummer (1 bis einschließlich 25 sind möglich) anzugeben. Soll mehr als eine Zeile gedruckt werden, so müssen aufeinanderfolgende Befehle das Zeilendrucken von oben nach unten erledigen. Für nichtinstallierte Einrichtungen erfolgt eine Befehlszurückweisung. Man beachte, daß dem Befehl „Drucken Zeile" irgendein anderer Befehl vorausgehen muß, um eine Karte in die Station nach der Stanzstation zu plazieren, da der ursprüngliche Kartenlauf nicht so verfährt.

Der Druckpuffer nimmt zwei Zeilen auf. Wird in einem Druckbefehl nun eine dritte Zeile angesprochen, so wird die erste im Puffer stehende Zeile gedruckt. Die letzten beiden Zeilen werden dann gedruckt, wenn der nächste Befehl für eine Kartenzuführung entdeckt wird.

Die Bedienungstasten sind die gleichen wie bei der IBM 3505 mit der Ausnahme, daß die Taste zum Haubenöffnen fehlt. Die Statusanzeigen sind ebenfalls die gleichen wie bei der IBM 3505.

In der Abb. 6.18. werden die Tasten und Lampen gezeigt, mit denen der Maschinenbediener vertraut sein muß. Einige der rückseitig beleuchteten Anzeichen unterscheiden sich von denen der IBM 3505. Dabei handelt es sich um die folgenden:

a) SCHNIPSELKASTEN
Der Schnipselkasten ist fast voll. Wenn die Anzeige OPERATOR RUF ebenfalls aufleuchtet, dann ist er gefüllt.

b) ABDKG OFFEN
Es ist sicherzustellen, daß der Lese- und Zuführungsmechanismus eingerastet ist. Danach kann START gedrückt werden.

c) START DRÜCKEN
Diese Anzeige leuchtet niemals allein auf. Sie zeigt an, daß durch Drücken von START die Operation fortgesetzt werden kann.

d) LETZT 3 KARTEN EINFÜGEN
Drei Karten sollten sich im Transportweg befinden. Sind es deren nur zwei, so ist die letzte Karte im Magazin nicht zugeführt worden.

e) OFFLINE
Die IBM 3525 ist nicht mit dem Kanal verbunden.

f) FEHLWAHL
Die Karte hat das geeignete Ablagefach verfehlt.

g) ABLAGE 3
Die Karte ist ins Ablagefach 3 eingelaufen.

h) SCHLECHTER DRUCK
Eine oder zwei der letzten beiden Karten sind möglicherweise schräg bedruckt worden.

Das Ablagefach 3 liegt innerhalb der IBM 3525. Wenn Karten nicht gelesen werden sollen, d. h. es sind keine Lesebefehle hinausgeschickt worden, so nimmt das System an, daß alle Karten zeichenfrei sind, d. h. ohne Lochungen vorliegen. Wird ein Stanzfehler entdeckt, so wird die verstanzte Karte ins Ablagefach 3 geführt und eine andere Lochkarte gestanzt. Wenn Karten gelesen werden, stoppt das System und die aufleuchtenden Anzeigen zeigen dem Maschinenbediener an, wie er sich bezüglich der fehlerhaften Karte zu verhalten hat.

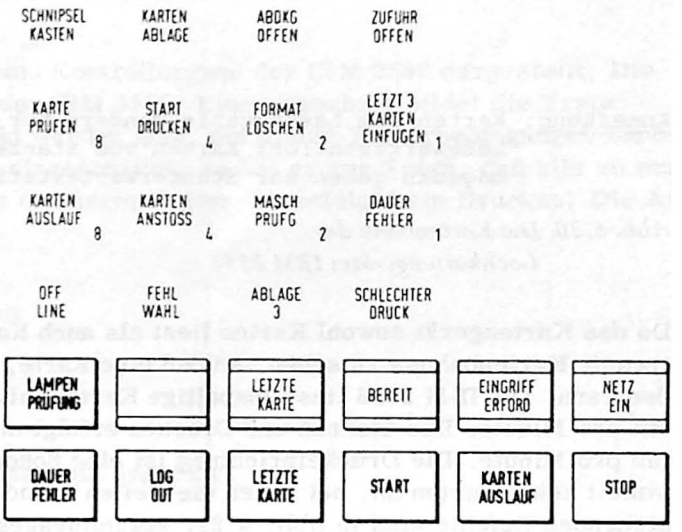


Abb. 6.18. Die Bedienungs- und Kontrollorgane der IBM 3525

6.7. Das Lochkartengerät IBM 2596

In Abb. 6.19. liegt die IBM 2596 fotografiert vor. Die Darstellung der Kartenbahn folgt in Abb. 6.20.

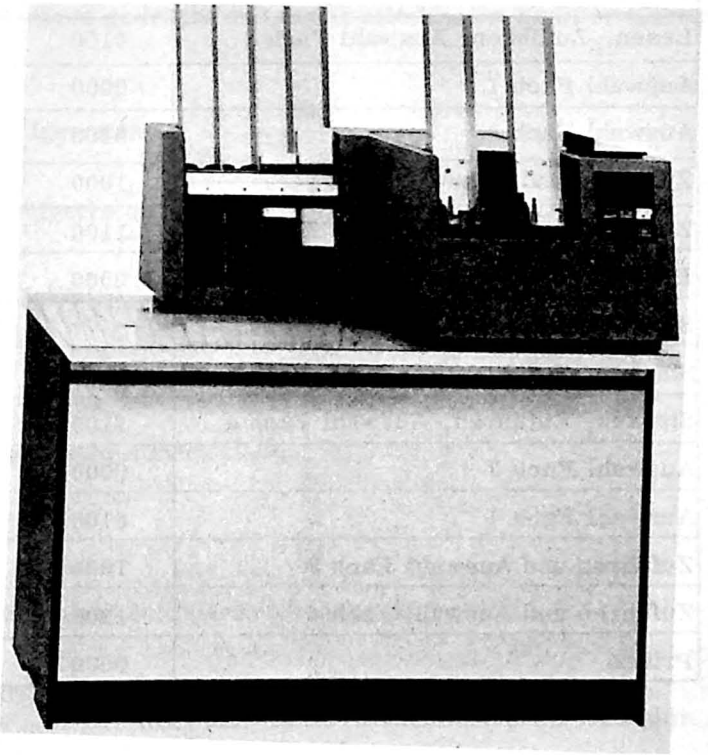
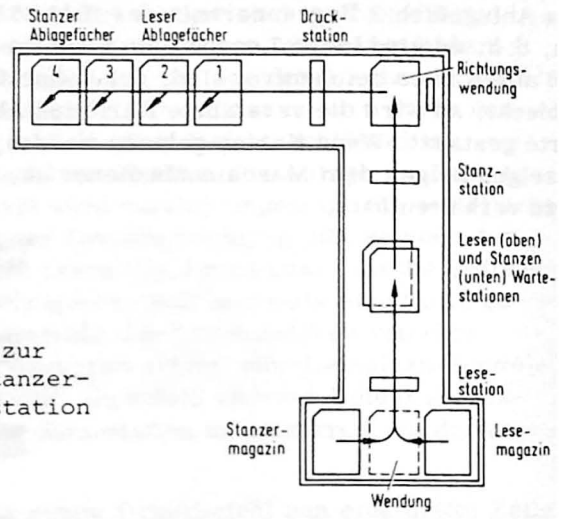


Abb. 6.19. IBM 2596



Anmerkung: Karten vom Lesemagazin wandern zur Lesewartestation; Karten vom Stanzermagazin gehen zur Stanzerwartestation

Abb. 6.20. Die Kartenbahn des Lochkartengerätes IBM 2596

Da das Kartengerät sowohl Karten liest als auch Karten stanzt, sind zwei logisch getrennte Kartenbahnen zu sehen, so daß eine Karte, die gelesen wird, nicht gestanzt werden kann. Die IBM 2596 liest 96spaltige Karten mit einer Geschwindigkeit von 500 Karten pro Minute. Das Stanzen und Drucken erfolgt mit einer Geschwindigkeit von 120 Karten pro Minute. Die Druckeinrichtung ist eine Sonderausstattung. — Die IBM 2596 nimmt keine Karten an, bei denen die Zeilen C und D abgelocht sind. Die zur Verfügung stehenden Befehle sind in Abb. 6.21. zusammengestellt.

Befehl	Befehlsschlüssel	
LESEN		
Lesen, Zuführen, Auswahl Fach 1	0000	0010
Lesen, Zuführen, Auswahl Fach 2	0100	0010
Auswahl Fach 1	0000	0011
Auswahl Fach 2	0100	0011
Zuführen und Auswahl Fach 1	1000	0011
Zuführen und Auswahl Fach 2	1100	0011
Prüfen	0000	0100
STANZEN		
Stanzen, Zuführen, Auswahl Fach 3	1000	0001
Stanzen, Zuführen, Auswahl Fach 4	1100	0001
Auswahl Fach 3	0000	0011
Auswahl Fach 4	0100	0011
Zuführen und Auswahl Fach 3	1000	0011
Zuführen und Auswahl Fach 4	1100	0011
Prüfen	0000	0100

Abb. 6.21. Die Befehlsschlüssel bei der IBM 2596

Ein Befehl, der die Fachauswahl ohne Zuführen veranlaßt, bewirkt, daß die Fachauswahl des nächsten Zuführbefehls ignoriert wird und stattdessen seine eigene Fachauswahl übernommen wird. Befehle, die Zuführen und Fachauswahl veranlassen, bewirken weder ein Lesen noch ein Schreiben.

Das Gerätestatusbyte besitzt das gleiche Aussehen wie das, das wir für die IBM 2501 kennengelernt haben (Abschnitt 6.2.). Das Prüfbyte benutzt nur die Bitstellen 0 bis 4, die die gleiche Bedeutung haben wie die des Prüfbytes bei der IBM 2501 (siehe Abschnitt 6.2.).

In der Abb. 6.22. sind die Bedienungs- und Kontrollorgane der IBM 2596 dargestellt. Die Tasten wirken fast genau so wie die bei der IBM 3505. Eine Ausnahme bildet die Taste KARTENAUSLAUF, die zweimal gedrückt werden muß, um einige Fehlerbedingungen zurücksetzen zu können. Ist der Druckschalter eingeschaltet, so hat er zur Folge, daß alle zu stanzenden Karten auch bedruckt werden, ist er ausgeschaltet, so erfolgt kein Drucken. Die Anzeigelampen besagen folgendes:

a) STANZER BEREIT
Die Stanzerzuführung ist betriebsbereit

b) LESER BEREIT
Die Leserzuführung ist betriebsbereit

c) STANZER ZUFÜHRUNG
Die letzte Karte wurde aus dem Stanzmagazin zugeführt

d) LESER ZUFÜHRUNG
Die letzte Karte wurde aus dem Lesermagazin zugeführt

e) LETZTE KARTE
Die Taste LETZTE KARTE wurde gedrückt

f) KARTENAUSLAUF
Die Taste KARTENAUSLAUF wurde gedrückt und damit alle Karten aus der Kartenbahn entfernt

g) STANZFEHLER
Beim Stanzen ist ein Fehler aufgetreten

h) DRUCKFEHLER
Beim Drucken ist ein Fehler aufgetreten

i) LESEFEHLER
Beim Lesen ist ein Fehler aufgetreten

j) ABLAGE
Volles Ablagefach

k) MASCH
Voller Schnipselkasten oder geöffnete Abdeckhaube

l) TEMP
Überhitzte Maschine

m) DATEN
Datenfehler: Das Zeichen gehört nicht zum 64-Zeichenvorrat

n) MAG
Fehler bei der Kartenzuführung vom Magazin

o) AUSG
Fehler in der Ausgangsleitung

p) ZUFÜHRUNG
Fehler in der Zuführung

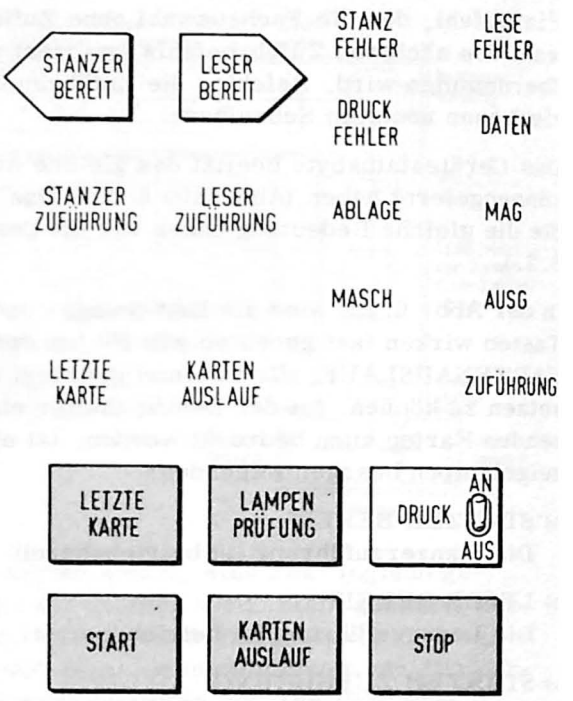


Abb. 6.22. Die Bedienungs- und Kontrollorgane der IBM 2596

6.8. Das Mehrfunktionskartengerät IBM 5425

Die Mehrfunktionskarteneinheit (MFKE) IBM 5425 sieht wie die IBM 2596 aus. Das rechte Kartenmagazin, von vorn gesehen, ist das Primärmagazin, das auf der linken Seite das Sekundärmagazin. Die Kartenbahn gleicht im großen und ganzen der der IBM 2596. Die von jedem Kartenmagazin aus einfließenden Karten können jedoch sowohl gelesen, gestanzt als auch bedruckt werden. Standardablagefächer sind 1 für das Primärmagazin und 4 für das Sekundärmagazin. Das Modell A1 liest 250 Karten pro Minute und stanzt 60; beim Modell A2 sind es 500 bzw. 120. Das Bedrucken von 1, 2 oder 3 Zeilen pro Karte hat keinen Einfluß auf die Stanzgeschwindigkeit, das Bedrucken einer 4. Zeile reduziert die Stanzgeschwindigkeit um ca. 20%.

Die IBM 5425 liest sowohl im Standardcode (256 Zeichen) als auch im sogenannten IPL-Code, einer 64 Zeichen umfassenden Untermenge.

Die Befehlsschlüssel sind in Abb. 6.23. zusammengefaßt.

Durch den Befehl „Ausgabe Stanzer" werden bis zu 96 Spalten zu einem Stanzpuffer gesendet, durch „Ausgabe Drucker" bis zu 128 Spalten zu einem Druckpuffer. Bei einem Fachauswahlbefehl kann das Ablagefach genannt werden, anderenfalls wird das Standardablagefach ausgewählt. Die Zuführungsbefehle sorgen für den Kartentransport. Die Karte an der Wartestation wird hinter die Stanz- und Druckstation bewegt und wird so gestanzt und bedruckt, wie es vorgesehen ist. Eine weitere Karte aus dem spezifizierten Magazin bewegt sich über die Lesestation hinweg bis zur Wartestation. Wird dabei der Befehl „Lesen und Zuführen" gegeben, so wird die Karte gelesen, sonst nicht. Alle Befehle bis zu und endend mit einem Zuführungsbefehl müssen sich auf dasselbe Kartenmagazin beziehen. Die Befehle können in beliebiger Reihenfolge gegeben werden, der Zuführungsbefehl muß stets der letzte sein.

Das Setzen der Fehlerbeseitigungsprozedur blockiert den Datentransport zu den Puffern für nachfolgende Ausgabefehle. Die Zurücksetzung richtet wieder die Normalprozedur ein. Nehmen wir z.B. einmal an, daß eine Karte gestanzt, jedoch noch nicht bedruckt

Befehl	Befehlsschlüssel	
Lesen und Zuführen	x0y0	0010
Ausgabe Stanzer	x000	0101
Ausgabe Stanzer, Stanzen und Zuführen	x000	0001
Ausgabe Drucker	x000	0101
Ausgabe Drucker, Drucken und Zuführen	x100	0001
Fachauswahl	xzzz	0011
Fachauswahl und Zuführen	xzzz	0111
Prüfen	0000	0100
Setzen der Fehlerbeseitigungsprozedur	0000	1111
Zurücksetzen der Fehlerbeseitigungs-prozedur	0000	1011

Abb. 6.23. Die Befehlsschlüssel bei der IBM 5425

Hierbei bedeuten:

x=0	Primärmagazin
x=1	Sekundärmagazin
y=0	Standardcode
y=1	IPL-Code
zzz=000	Standardablagefach 1 oder 4
zzz=001	Ablagefach 1
zzz=010	Ablagefach 2
zzz=011	Ablagefach 3
zzz=100	Ablagefach 4

wurde und daß ein Fehler zur Betätigung des Kartenauslaufes zwang. Dann könnten wir folgende Befehlsfolge veranlassen:

— Setzen der Fehlerbeseitigungsprozedur
— Zuführen
— Ausgabe Drucker und Zuführen
— Ausgabe Stanzer
— Ausgabe Drucker und Zuführen
— Zurücksetzen der Fehlerbeseitigungsprozedur

Man beachte hierbei, daß es zwei Druckpuffer gibt, da das Bedrucken einer Karte weiter geht, während der Befehl „Ausgabe Drucker" für die nachfolgende Karte ausgeführt wird. Durch Blockierung des Datentransportes wird die erste Karte mit ihrem im Puffer aufbewahrten Inhalt bedruckt und die zweite Karte wird korrekt gestanzt und bedruckt.

Das Gerätestatusbyte weist als bedeutungsvolle Bitstellen nur die folgenden auf:

— In Arbeit
— Kanalende
— Geräteende
— Gerätefehler

Das Prüfbyte 0 enthält:

a) Bitstelle 0 — Befehlszurückweisung
Ungültiger Befehl oder Befehlsreihenfolge

b) Bitstelle 1 — Eingriff erforderlich
Kartenmagazin nicht bereit; MFKE nicht betriebsbereit; Magazinfehler, Zuführungsfehler

c) Bitstelle 2 — nicht benutzt

d) Bitstelle 3 — Ausrüstungsfehler

e) Bitstellen 4 und 5 — nicht benutzt

f) Bitstelle 6 — Nichtverfügbarkeit
Die Wartestation war bei Hinausschickung eines Ausgabebefehles leer (sie ersetzt die Geräte-Ausnahmebedingung bei Geräten mit einer Zuführung)

g) Bitstelle 7 — nicht benutzt

Im Prüfbyte 1 werden Maschinenfehler festgehalten. Das Prüfbyte 2 hält die Lage der Karten in der Kartenbahn fest. Seine einzelnen Bitstellen bedeuten:

a) Bitstellen 0 und 1 — nicht benutzt
b) Bitstelle 2 — Karte vom Primärmagazin in Wartestation
c) Bitstelle 3 — Karte vom Sekundärmagazin in Wartestation
d) Bitstelle 4 — nicht benutzt
e) Bitstelle 5 — Magazinzyklus nicht komplett
Der Zuführungsbefehl wurde zwar gestartet, doch die Karte hat das Magazin nicht verlassen
f) Bitstellen 6 und 7 — Zähler
Anzahl der Karten zwischen der Wartestation und den Ablagefächern

Die Prüfbytes 3, 4 und 5 dienen Diagnostikzwecken. In den Bytes 6 bis 10 werden Informationen über die letzten 5 Befehle aufbewahrt. Dabei befindet sich die Information über den letzten Befehl im Prüfbyte 6, die über den vorletzten im Prüfbyte 7 usw. Auf der Bitstelle 0 dieser Bytes steht eine 1, wenn der Befehl für die Sekundärzuführung galt. Bitstelle 1 enthält eine 1, wenn mehr als 96 Bytes gedruckt wurden (4 Druckzeilen). Die Bitstellen 2 und 3 enthalten 00, wenn Ablagefach 1, 01, wenn Ablagefach 2, 10, wenn Ablagefach 3 und 11, wenn Ablagefach 4 angesprochen waren. Die Bitstelle 4 steht auf 1, wenn der Stanzpuffer geladen wurde. Der Bitstelle 5 ist eine 1 zugewiesen, wenn der betreffende Befehl eine Zuführung einleitete. Die Bitstelle 6 steht auf 1, wenn der Druckpuffer geladen wurde und Bitstelle 7, wenn eine Karte gelesen wurde.

Die Bedienungs- und Kontrollorgane sind in Abb. 6.24. gezeigt. Es gibt folgende Tasten:

a) STOP
Versetzt die MFKE in den nicht betriebsbereiten Zustand

b) START
Versetzt die MFKE in den betriebsbereiten Zustand. Ebenso wird jedes Magazin betriebsbereit gemacht, das Karten enthält. Es werden durch das Drücken dieser Taste jedoch keine Karten zugeführt

c) KARTENAUSLAUF
Es laufen die Karten aus, deren Magazin leer ist. Um beide Magazine zu leeren, ist diese Taste zwei Mal zu drücken.

Die Lampenfelder PRIMÄR BEREIT und SEKUNDÄR BEREIT zeigen an, welche Zuführungen betreibsbereit sind.

Die rückseitig beleuchteten Anzeigen verweisen auf folgende Bedingungen:

a) PRI
Die zuletzt zugeführte Karte stammt vom Primärmagazin

b) SEK
Die zuletzt zugeführte Karte stammt vom Sekundärmagazin

c) LES
Die zuletzt gelesene Karte verursachte einen Zuführungsfehler

d) MAG
Die Karte wurde nicht zugeführt

e) AUSLAUF
KARTENAUSLAUF drücken, um Fehler zu beseitigen

f) ABLAGE
Volles Ablagefach; entleeren und START drücken

g) SCHNIPSEL
Schnipselkasten voll; entleeren und START drücken

h) ABDKG
Geöffnete Abdeckhaube; schließen und START drücken

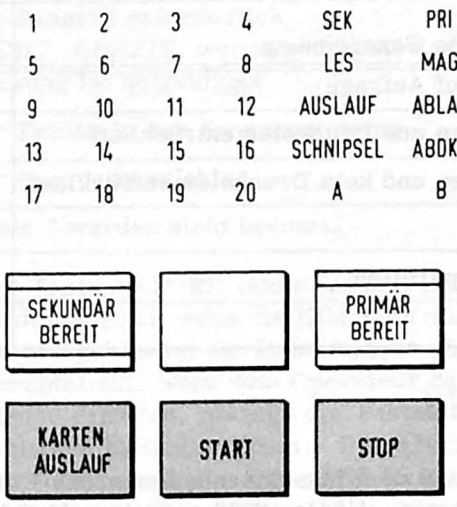


Abb. 6.24. Bedienungs- und Kontrollorgane der IBM 5425

6.9. Der Konsoldrucker IBM 3215 mit Tastatur

Die IBM 3215 besteht aus einer schreibmaschinenähnlichen Tastatur, einigen Steuertasten und -lampen sowie aus einem Druckmechanismus, der ziemlich ähnlich einer Schreibmaschine arbeitet. Die Zeichen werden als Punktraster gedruckt, wobei Drahtstifte auf das Farbband einhämmern. Jedes Zeichen wird aus einem Druckelement gebildet, das sich aus 7 vertikal übereinander angeordneten Druckpositionen in 7 Spalten zusammensetzt. Die Zeichendichte beträgt 10 Zeichen pro Zoll (3,93 Zeichen/cm). Sechs Zeilen verteilen sich auf ein Zoll, d.h. 2,36 Zeilen kommen auf 1 cm. Die IBM 3215 ist für kurze Nachrichten an den Systembediener (Konsoloperateur) und für das Eintasten seiner Antworten bestimmt. Die IBM 3215 wird gewöhnlich an eine Adresse im Spektrum 01F bis 009 angeschlossen.

In der Abb. 6.25. ist die Tastatur schematisch dargestellt. Die insgesamt verfügbaren 89 Zeichen schließen die bei der IBM 029 vorhandenen 64 Zeichen sowie die 26 Kleinbuchstaben ein; es fehlt der Code 0-2-8 der IBM 029.

Abb. 6.25. Schematische Übersicht über die Tastatur und die Steuerungsorgane der IBM 3215

Die Befehlsschlüssel für die IBM 3215 finden wir in der Abb. 6.26.

Engl. Bezeichnung	*Deutsche Bezeichnung*	*Befehls-schlüssel*
Read Inquiry	Lesen auf Anfrage	0A
Write and Carrier Return	Schreiben und Druckelementrücklauf	09
Write and no Carrier Return	Schreiben und kein Druckelementrücklauf	01
Sense	Prüfen	04
Nop (no Operation)	Keine Operation	03
Alarm	Alarm	0B

Abb. 6.26. Die Befehlsschlüssel bei der IBM 3215

Die Operation „Lesen auf Anfrage" erlaubt dem Maschinenbediener das Eintasten irgendwelcher Zeichen. Die Operation endet damit, daß der Zähler im Kanalbefehlswort (CCW) auf Null abgebaut ist, oder daß der Bediener eine der Tasten UNG (ungültig) oder ENDE drückt. Die Schreibbefehle bewirken, daß die Daten aus dem Hauptspeicher mit einer

Geschwindigkeit von 85 Zeichen pro Sekunde gedruckt werden. Die Schreiboperationen enden, wenn der Zähler im CCW auf Null heruntergegangen ist oder wenn der Bediener die ENDE-Taste drückt. Die Leseoperation sowie die Schreiboperation mit Druckelementrücklauf enden stets mit einem automatischen Druckelementrücklauf. Bei der anderen Schreiboperation beharrt das Druckelement auf der Position, die eine Stelle rechts vom letzten gedruckten Zeichen liegt. Die über die Tastatur eingegebenen Zeichen werden im gleichen Augenblick gedruckt, in dem sie zum Hauptspeicher gesendet werden. Der Befehl „Alarm" ruft ein etwa $1\frac{1}{2}$ Sekunden andauerndes akustisches Signal hervor.

Die Bitstellen des Gerätestatusbytes haben folgende Bedeutung:

a) Bitstelle 0 — Achtung (Attention)
Dieser Bitstelle wird eine 1 zugewiesen, wenn die Taste ANFORD (Anforderung) gedrückt wird

b) Bitstelle 1 und 2 — Nicht benutzt

c) Bitstelle 3 — In Arbeit

d) Bitstelle 4 — Kanalende

e) Bitstelle 5 — Geräteende

f) Bitstelle 6 — Gerätefehler
Diese Bitstelle enthält immer dann eine 1, wenn nicht alle Bits des Prüfbytes gleich 0 sind. Diese Bedingung tritt nur während der Anfangsansteuerung auf, jedoch nicht während einer Befehlsausführung

g) Bitstelle 7 — Geräte-Ausnahmebedingung
Diese Bitstelle wird auf 1 gesetzt, wenn ein Lesen durch das Drücken der Taste UNG (ungültig) beendet wird. Dieser Tastenanschlag signalisiert in der Regel, daß bei der Eingabe ein Tippfehler gemacht wurde. Gewöhnlich ist das Programm so gestaltet, daß der Operateur die Eingabe wiederholen kann

Das Prüfbyte besitzt folgenden Aufbau:

a) Bitstelle 0 — Befehlszurückweisung
Ein nicht definierter Befehlsschlüssel wurde von der IBM 3215 empfangen

b) Bitstelle 1 — Eingriff erforderlich
Die Taste NICHT BEREIT wurde gedrückt, das Druckpapier geht zur Neige oder die Abdeckung ist aufgeklappt

c) Bitstelle 2 — Fehler in der Ausgangsleitung

d) Bitstelle 3 — Ausrüstungsfehler

Die Bitstellen 4 bis 7 werden nicht benutzt.

Beim Drücken der Taste ANFORD (Anforderung) leuchtet die Lampe ANFORD ANST (anstehende Anforderung) auf; wenn die IBM 3215 nicht betriebsbereit ist, kommt es zum Setzen des Achtungsbits im Gerätestatusbyte und die Lampe verlöscht. Die Lampe FORTFAHREN leuchtet auf, wenn dem Operateur das Eintasten erlaubt ist. Ihr Leuchtzustand bleibt solange erhalten, solange die Tastatur nicht gesperrt ist und damit der Operateur das Eintasten fortsetzen kann. Der Alarmbefehl besorgt neben dem deutlichen Geräusch auch das Aufleuchten der Lampe ALARM. Die Taste zum Löschen des Alarms (ALARM RÜCKS — Alarm rücksetzen) schaltet gleichzeitig dieses Licht wieder aus. Durch das Drücken der Taste UNG (ungültig) werden nur Leseoperationen abgeschlossen; zur gleichen Zeit wird die Geräte-Ausnahmebedingung gesetzt. Das Anschlagen der Taste

NICHT BEREIT versetzt die IBM 3215 in den nicht betriebsbereiten Zustand, der durch das Drücken der Taste BEREIT wieder aufgehoben werden kann. Die Lampe EINGR NOTW (Eingriff notwendig) leuchtet immer dann auf, wenn die Bitstelle 1 des Prüfbytes (Eingriff erforderlich) eine 1 enthält. Die Taste ÄND/ANZEIGE (Änderung/Anzeige) und die Lampe ÄND/ANZ MODUS (Änderung/Anzeige-Modus) kommen immer dann zum Zuge, wenn sie der Operateur zur Änderung des Inhalts von Speicherstellen oder Register oder zum Drucken ihrer Inhalte benutzt. Die Benutzung dieser Einrichtung ist von Modell zu Modell verschieden.

Der Konsoldrucker mit Tastatur IBM 3210 gleicht in seiner Funktion dem Konsoldrucker IBM 3215. Die Druckgeschwindigkeit ist jedoch auf 15,5 Zeichen/Sekunde reduziert, was durch den verwendeten Schreibmechanismus (Kugelkopfschreibmaschine) bedingt ist.

6.10. Die Konsole IBM 3125

Die Konsole IBM 3125 besteht aus einer Tastatur, deren Belegung schematisch in Abb. 6.27. gezeigt ist, einem Bildschirm (Kathodenstrahlröhre). Auf dem Bildschirm können bis zu 16 Zeilen mit je 56 Zeichenstellen erscheinen. An diese Konsole kann außerdem ein Drucker des Typs IBM 5213 angeschlossen werden, gewissermaßen als Druckkomponente, durch die lesbare Kopien (Hardcopies) von einigen oder allen Informationen hergestellt werden können, die auf dem Bildschirm auftauchen. Beide Geräte weisen nur eine Geräteadresse auf, der Drucker funktioniert nicht unabhängig von der Bildschirmkonsole. Das Datensichtgerät IBM 3125 dient nicht nur als gewöhnliche Konsole wie die IBM 3215, es sind vielmehr in ihm auch Anzeiger und andere Kontrollorgane eingebaut, die sonst an der Zentraleinheit (ZE) zu finden sind. Die Konsole IBM 3125 wird u.a. an das Modell 125 des Systems IBM /370 angeschlossen.

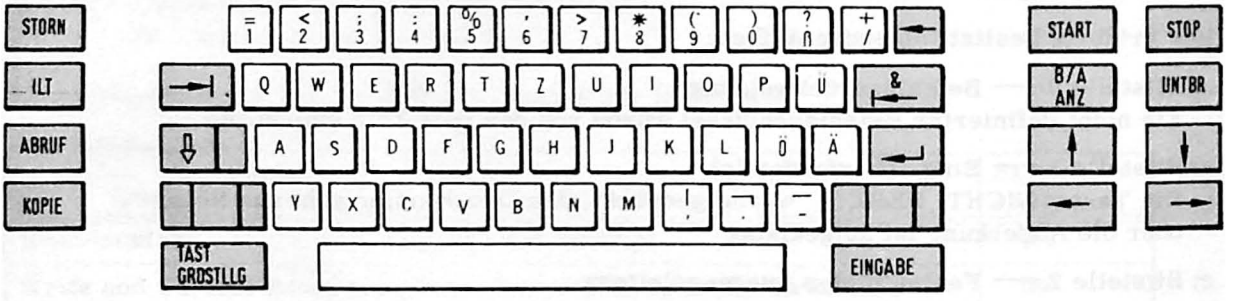


Abb. 6.27. Belegung der Tastatur der Bildschirmkonsole IBM 3125

Die untersten vier Zeilen auf dem Bildschirm zeigen Informationen über den Status der ZE an. Sie können nicht vom Programm kontrolliert werden. Tatsächlich bleiben also nur 12 Zeilen zur Anzeige von Informationen übrig. Das entspricht einer darstellbaren Zeichenmenge von maximal 12·56 gleich 672 Zeichen auf dem Bildschirm. Die Zeichenstellen sind fortlaufend von 1 bis 672 numeriert. Zusätzlich existiert noch eine Stelle 0 unmittelbar links vor der ersten Zeichenstelle der ersten Zeile. Auf jeder Zeichenstelle kann

— ein Datenzeichen oder
— ein Attributzeichen

erscheinen. Die Datenzeichen sind die gleichen wie bei der IBM 3215 mit der Ausnahme, daß keine Kleinbuchstaben angezeigt werden. Die Kathodenstrahlröhre wird über einen 673 Bytes fassenden Puffer versorgt. Die gesamte Anzeigefläche kann in ein oder mehrere Felder unterteilt werden. Jedes Feld beginnt mit einem Attributzeichen und setzt sich bis zum nächsten Attributzeichen fort bzw. endet mit dem 672. Byte. Einsen auf den Bitstellen der Attributzeichen haben die in Abb. 6.28. genannte Bedeutung.

Bitstelle	*Bedeutung*
0	Drucken: Das Feld kann über die IBM 5213 ausgegeben werden
1	enthält immer 1
2	Schutzbit: Der Maschinenbediener kann den Inhalt des Feldes nicht ändern
3 bis 6	nicht benutzt
7	Änderungsdaten: Durch den Befehl „Modifiziertes Lesen" kann der Feldinhalt gelesen werden

Abb. 6.28. Aufbau der Attributzeichen

Obgleich Attributzeichen Speicherplatz belegen, werden sie auf dem Bildschirm nicht angezeigt. Außerdem gibt es ein Positionsanzeigesymbol [1], ein schmaler Balken, der auf dem Bildschirm diejenige Stelle bezeichnet, auf die der Maschinenbediener das nächste Datenzeichen eintasten kann. Sollte der Operateur versuchen, Daten in ein geschütztes Feld einzutasten, so bleibt die Tastatur verriegelt. Erst durch das Drücken der Taste TAST GRDSTLLG wird die Tastatur wieder entriegelt.

Acht Tasten sind vorgesehen, um die Positionsanzeige zu bewegen. Die mit

beschrifteten Tasten führen beim Anschlag die Positionsanzeige nach oben, unten, nach links und nach rechts bzw. auf die erste Stelle eines Feldes, das durch ein Attributzeichen eingeleitet wird, bei dem die Schutzbitstelle auf 0 steht. Das Drücken der Taste mit dem Symbol ⏎ führt die Positionsanzeige in die linke untere Ecke, von der aus das erste Byte gesucht wird, das hinter einem Attributzeichen ohne Schutzbit erscheint. Alle Bewegungen der Positionsanzeige erfolgen zyklisch, d.h. sie gehen von der Zeile 12 wieder zur Zeile 1 zurück. Alle Tasten, die in Abb. 6.27. durch ein geschwärztes Dreieck in der rechten oberen Ecke markiert sind, sind Tasten mit Dauerfunktion, d.h. werden sie niedergedrückt und nicht nur angeschlagen, so führt das dazu, daß die durch sie ausgelöste Funktion wiederholt wird.

Einige Systemoperationen verriegeln zeitweise die Tastatur. In solchen Fällen muß der Operateur bis zum Abschluß der Operation warten. Ein Anzeiger auf der Zeile 13 zeigt den Zustand der Tastatur an.

Die Tasten START, STOP, B/A ANZ und UNTBR ersetzen die normalen sonst an der Zentraleinheitskonsole angebrachten Tasten. Die Taste ILT (Inlinetest) wird nur vom Wartungspersonal bedient. Die Taste ABRUF wird nur mit Sonderausrüstung gebraucht, die dann den Bildschirm genau so agieren läßt, wie eine IBM 3215.

Ein sogenanntes Achtungsidentifizierungsbyte zeigt den Zustand der Tastatur an. Normalerweise ist sein Inhalt 60 (sedezimal). Das Drücken der Taste EINGABE weist diesem Byte den Wert 70 (sed) zu und setzt gleichzeitig die Achtungsbitstelle im Gerätestatus-

1) Im angloamerikanischen Sprachraum als „Cursor" bezeichnet.

byte auf 1. Das Anschlagen der Taste STORN (ungültig) bringt dieses Byte auf den Wert 6E (sed) und setzt die Achtungsbitstelle an.

Die Befehlsschlüssel sind in Abb. 6.29. aufgeführt.

Engl. Bezeichnung	*Deutsche Bezeichnung*	*Befehlsschlüssel*
Write	Schreiben	01
Erase/Write	Löschen/Schreiben	05
Erase all Unprotected	Löschen aller ungeschützten Felder	0F
Read modified	Modifiziertes Lesen	06
Sense	Prüfen	04
Nop	Keine Operation	03

Abb. 6.29. Die Befehlsschlüssel bei der IBM 3125

Durch den Schreibbefehl werden 1 bis 2048 Bytes in einen Zwischenpuffer geschickt. Das erste (oder auch einzige) Zeichen ist das Schreibsteuerzeichen. Die restlichen Bytes, falls solche vorhanden sind, enthalten Steuerinformationen und die anzuzeigenden Daten. Die verwendeten Steuerzeichen sehen wie folgt aus:

Bitstellen	Bedeutung
0 bis 3	ohne Bedeutung
4	Starten Drucker
5	Akustischer Alarm
6	Tastaturverfügbarkeit herstellen
7	Zurücksetzen der modifizierten Datenbits auf Null

Eine 1 auf der Bitstelle 4 bewirkt, daß die Inhalte aller auf dem Bildschirm ausgegebenen Felder, deren Druckbit an ist, auf der IBM 5213 gedruckt werden. Der Alarm ähnelt dem bei der IBM 3215. Eine 1 auf der Bitstelle 6 bewirkt, daß die Tastatur entriegelt wird und außerdem, daß das Achtungsidentifizierungsbyte zurück auf 60 (sed) gesetzt wird. Die letzte Bitstelle setzt bei einer enthaltenen 1 alle modifizierten Datenbitstellen in allen Feldern auf Null.

Innerhalb der Daten können drei Arten von Gerätesteuerinformationen auftreten. Das Setzen der Pufferadresse erfordert die Belegung von drei Bytes; im ersten ist 11 (sed)

Bytes	1								2															
Bitstellen	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
Wert für die Adresse	unbenutzt								nicht benutzt			1024	512	256	128	64	nicht benutzt		32	16	8	4	2	1
									← Pufferadresse →															

Abb. 6.30. Bedeutung der Bitstellen in der Pufferadresse

enthalten. Die nächsten beiden beinhalten die Schirmadresse, die nicht größer als 672 sein darf. Die Werte, die den einzelnen Bitstellen dieser Bytes zugewiesen sind, haben die in Abb. 6.30. dargestellte Bedeutung.

Jedem Attributbyte geht ein Byte mit dem sedezimalen Inhalt 1D voraus; das Attributbyte folgt auf der nächsten Stelle. Durch eine Adreßsetzungsfolge kann die Stellung eines nachfolgenden Attributbytes festgelegt werden. Der sedezimale Wert 13 setzt die Positionsanzeige auf die nächste verfügbare Stelle. Es gibt keine Beschränkung hinsichtlich der Zahl von Adreß-, Attribut- und Positionsanzeige-Steuerbytes, die gesamte Zeichenzahl darf freilich die Kapazität des Zwischenpuffers, der 2048 Zeichen aufnehmen kann, nicht überschreiten. Geht dem ersten Datenbyte keine Folge von Adreßbytes voraus, so wird die Stelle versorgt, auf der die Positionsanzeige gerade steht. Die Daten werden im Zwischenpuffer analysiert und dann dazu benutzt, um die Inhalte der angesprochenen (adressierten) Bytes im Bildschirmpuffer zu ersetzen. Mit Ausnahme dieser Bytes bleibt der übrige Inhalt des Bildschirmpuffers erhalten. Da keine zyklische Adressierung für den Bildschirm vorhanden ist, d.h. der Übergang von Byte 672 zu Byte 0 fehlt, muß der Platz mit der Adresse 0 ein Attributbyte aufnehmen.

Der Befehl „Löschen/Schreiben" bewirkt bei seiner Ausführung, daß der gesamte Bildschirmpuffer gelöscht wird; anschließend erfolgt ein Schreiben. Als Teil der Löschoperation wird die Positionsanzeige auf die obere linke Ecke des Bildschirms geführt.

Der Befehl „Löschen aller ungeschützten Felder" bewirkt keinen Datentransport. Geschützte Felder werden von dieser Operation nicht betroffen. Ungeschützte Felder werden gelöscht und die Bitstelle 7 ihres Attributzeichens auf 0 gesetzt (siehe Abb. 6.28.). Durch diesen Befehl wird außerdem dem Achtungsidentifizierungsbyte der Grundwert 60 (sed) zugewiesen, wodurch gleichzeitig auch die Sperre der Tastatur aufgehoben wird. Außerdem wird die Positionsanzeige auf die am weitesten links gelegene Stelle des ersten gelöschten Feldes geführt.

Der Befehl „Modifiziertes Lesen" überträgt die modifizierten Daten wie folgt:

1) Wurde die Taste UNG (ungültig) gedrückt, so wird nur das Achtungsidentifizierungsbyte gesendet.

2) In allen anderen Fällen folgt dem Achtungsidentifizierungsbyte die Adresse der Positionsanzeige (zwei Bytes). Daran schließen sich für jedes Feld, dessen Bitstelle 7 des zugehörigen Attributzeichens (siehe Abb. 6.28.) eine 1 enthält, folgende Informationen an:

   (a) Pufferadresse (zwei Bytes)
   (b) Datenzeichen des Feldes (ohne Attributbyte)

Besprechen wir nun das Gerätestatusbyte. Seine Bitstellen haben folgende Bedeutung:

a) Bitstelle 0 —— Achtung (Attention)
   Dieser Bitstelle wird eine 1 zugewiesen, wenn eine der Tasten EINGABE oder UNG gedrückt wird.

b) Bitstelle 1 —— nicht benutzt

c) Bitstelle 2 —— Steuereinheitenende
   Wenn die Konsole für ZE-Funktionen benötigt wird, gilt das Steuergerät als in Arbeit befindlich. Steuereinheitenende wird deshalb signalisiert, wenn das Gerät verfügbar wird

d) Bitstelle 3 —— In Arbeit

e) Bitstelle 4 —— Kanalende

f) Bitstelle 5 — Geräteende
Normalerweise ist die Tastatur entriegelt, deshalb ist Geräteende nur auf die Fähigkeit bezogen, einen anderen Befehl ausführen zu können, nicht auf die Aktivität an der Tastatur

g) Bitstelle 6 — Gerätefehler
Dieser Bitstelle wird eine 1 zugewiesen, wenn irgendeine Bitstelle des Prüfbytes eine 1 enthält

h) Bitstelle 7 — Geräte-Ausnahmebedingung
Dieser Bitstelle wird nur dann eine 1 zugewiesen, wenn das Drucken durch Drücken einer der Tasten MODUS AUSW oder STOP angehalten wird

Die einzelnen Bitstellen des Prüfbytes haben folgende Bedeutung:

a) Bitstelle 0 — Befehlszurückweisung
Verwendung eines nicht definierten Befehlsschlüssels

b) Bitstelle 1 — Eingriff erforderlich
Das Setzen dieser Bitstelle rührt nur von der IBM 5213 her

c) Bitstelle 7 — Operationsprüfung
Folgende Fehlerquellen verursachen das Setzen einer 1 in diese Bitstelle:

— Pufferadresse größer als 672
— Platz 0 enthält kein Attributzeichen
— Die Befehle „Schreiben" und „Löschen/Schreiben" greifen auf eine unvollständige Zeichenmenge bezüglich Pufferadressen und Attributzeichen zurück

6.11. Zeilendrucker

Die Daten auf Papier zu drucken, von dem sie leicht abgelesen werden können, ist sicherlich in vielen Fällen eine zufriedenstellendere Lösung, als sie in Karten zu stanzen. Das Drucken von Daten ist auch über die Konsolschreibmaschine möglich, doch die geringe Geschwindigkeit einer Schreibmaschine läßt sie nur zur Ausgabe von kurzen und selten vorkommenden Nachrichten geeignet erscheinen. Die Zeilendrucker, die an die IBM /370 angeschlossen werden können, drucken mit einem Male ein ganze, aus 96 bis 150 Zeichen bestehende Zeile. Der Drucker IBM 1443, Modell N1, druckt 200 bis 600 Zeilen/Minute (abgekürzt Z/m), wobei jede Zeile 120, bei Zahlung von Sonderkosten sogar 144 Druckstellen umfaßt. Die IBM 1403, Modell 2, druckt zwischen 143 bis 1285 Z/m, wobei jede Zeile aus 132 Druckstellen besteht. Die IBM 1403, Modelle 3 und N1, drucken mit einer Geschwindigkeit zwischen 309 Z/m und 1400 Z/m; auch bei ihnen umfaßt jede Druckzeile genau 132 Druckstellen. Der Drucker IBM 1443 enthält eine eigene Steuerung; die Drukker des Typs IBM 1403 müssen dagegen über ein Steuergerät angeschlossen werden.

Die IBM 1443 verwendet zum Drucken eine lange Typenstange, auf der entweder 13, 39, 52 oder 63 verschiedene Zeichen untergebracht sind. Die Typenstange bewegt sich abwechselnd von links nach rechts und dann von rechts nach links. Elektromagnetische Hämmer schlagen während der Bewegung gegen die Stange, wodurch die entsprechenden Zeichen gedruckt werden. Die Druckgeschwindigkeit hängt von der Anzahl der auf der Typenstange untergebrachten Zeichen wie folgt ab:

Anzahl der Zeichen	Z/m
13	600
39	300
52	240
63	200

Die 52 Zeichen enthaltende Typenstange ist die Stange, mit der der Drucker IBM 1443 standardmäßig ausgerüstet wird. Dabei sind verschiedene Zeichenanordnungen möglich.

Die IBM 1403, Modell 2, verwendet eine Druckkette, ähnlich einer Fahrradkette, auf der 240 Typen montiert sind. Verwendet man die Standardanordnung von 48 Zeichen, so wiederholt sich jedes Zeichen gerade fünf Mal; auf diese Weise wird eine Druckgeschwindigkeit von 600 Z/m erreicht. Wie bei der IBM 1443 sind auch hier verschiedene Zeichenanordnungen verfügbar. Das Modell 7 ist dem Modell 2 ähnlich, jedoch weist es eine Zeilenbreite von nur 120 Druckstellen auf; außerdem wird das Papier in einigen Fällen langsamer bewegt.

Die IBM 1403, Modelle 3 und N1, sind Gliederdrucker, wodurch sich die Geschwindigkeit von 600 Z/m auf 1100 Z/m erhöht (beim 48-Typen-Satz).

Die IBM 5203, Modell 3, benutzt wie die IBM 1403-N1 zum Drucken Glieder. Die Druckgeschwindigkeit beträgt 300 Z/m, die Druckzeile umfaßt je nach Ausrüstung 96, 120 oder 132 Druckstellen.

Die IBM 3203, Modell 1, ist ebenfalls mit Druckgliedern ausgestattet. Die Druckgeschwindigkeit beträgt 600 Z/m. Die Druckzeile besteht aus 132 Druckstellen. Beim Modell 2 dieses Druckertyps ist die Geschwindigkeit auf 1200 Z/m erhöht.

Die IBM 3211, Modell 1, benutzt 432 Druckglieder, was sie auf eine Geschwindigkeit von 2000 Z/m bei 132 oder 150 Druckstellen/Zeile bringt.

Mit Ausnahme der IBM 1403-7 steht bei allen anderen Druckern ein Universaltypensatz [1] zur Verfügung. Er ermöglicht es dem Bedienungspersonal, die Druckeinrichtungen (Ketten oder Glieder) selbst zu montieren. Die verschiedenen Druckeinrichtungen können dabei gänzlich verschiedene Zeichenmengen umfassen. Ein UCS-Puffer wird mit den EBCDIC-Abbildern der auf der Druckeinrichtung vorhandenen Zeichen geladen. Bei einigen Zeichenmengen werden nur die gewöhnlich benutzten Zeichen ständig wiederholt. Weniger häufig benutzte Zeichen brauchen nur einmal zu erscheinen. Beim Drucken solcher Nichtvorzugszeichen sinkt naturgemäß die Druckgeschwindigkeit herab, oft gerade auf die Hälfte der regulären. Beim Drucken von Zeilen mit Vorzugszeichen wird die normale Geschwindigkeit nahezu erreicht. Dieser Kompromiß zwischen der Fähigkeit, viele verschiedene Zeichen drucken zu können, und weitgehender Beibehaltung der normalen Druckgeschwindigkeit beim Druck häufig benutzter Zeichen, ist sicher begrüßenswert. Die Druckgeschwindigkeit hängt nämlich davon ab, wieviel Mal jedes Zeichen im Typensatz vorkommt.

Die Papierbewegung kann zum Teil durch ein Lochband für die Vorschubsteuerung gesteuert werden. Dieses Papierband (neuerdings kann es auch aus einem Kunststoff bestehen) besitzt die gleiche Länge wie das zu bedruckende Formular. In 12 verschiedene Kanäle können Löcher zur Vorschubsteuerung eingelocht werden. Man betrachte dazu die Abb. 6.31. Die Löcher werden in das Band so eingelocht, daß sie mit den auf dem Formular hervorzuhebenden bzw. besonders anzusteuernden Druckzeilen übereinstimmen. Nach dem Ablochen wird das Band zu einer endlosen Schleife zusammengeklebt und auf die Vorschubsteuerungseinrichtung des Druckers aufgespannt. Danach wird es so positioniert, daß die erste Druckzeile des Formulars dem ersten Loch auf dem Band entspricht. Nach dieser Ausrichtung bewegen sich Band und Formular zu-

1) Im angloamerikanischen Sprachraum als „Universal Character Set", abgekürzt UCS, bezeichnet

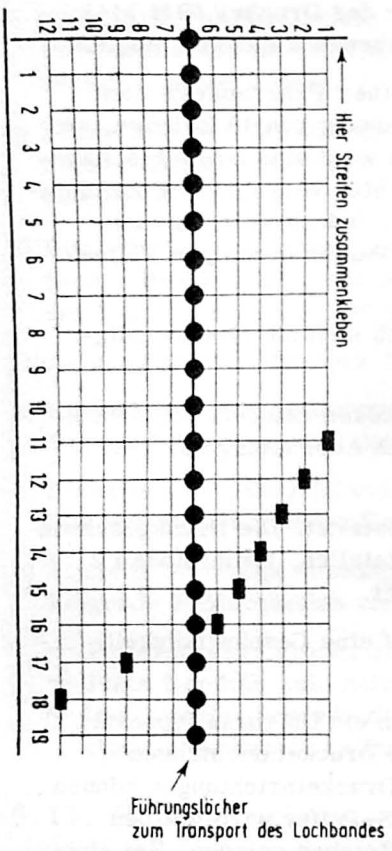


Abb. 6.31. Lochband zur Vorschubsteuerung

sammen mit übereinstimmenden Geschwindigkeiten. Im allgemeinen wird durch eine Lochung im Kanal 1 die erste Druckzeile markiert und durch eine Lochung im Kanal 12 die letzte Druckzeile. Wenn das Formular sehr kurz ist, können selbstverständlich die Lochungen im Vorschubsteuerungsband wiederholt werden, wodurch das Band zwei-, drei- oder viermal so lang wird als das Formular. Einer Umdrehung des Lochbandes entspricht somit die Zuführung der entsprechenden Anzahl von Formularen.

Eine Lochung im Kanal 9 wird oftmals dazu verwendet, um eine bzw. Summenzeile eines Formulars zu kennzeichnen; mitunter wird aber auch durch eine solche Lochung die letzte Druckzeile eines Formulars markiert.

Die moderne Praxis verzichtet jedoch auf Techniken, die das Einlegen von Speziallochbändern für jeden Job erfordern. Bei einigen Druckern, z.B. der IBM 3211 gibt es deshalb solche physischen Lochbänder überhaupt nicht mehr. Stattdessen wird ein Formularsteuerungspuffer[1]) mit einem Abbild des Lochbandes geladen. Damit sind schnellere Formularwechsel bei zugleich größerer Zuverlässigkeit gegeben.

Alle Drucker können auf eine Zeilendichte von 6 oder 8 Zeilen pro Zoll[2]) eingestellt werden. Bei einigen Modellen mit dem Formularsteuerungspuffer wird die Einstellung in den Puffer gestellt.

Für die Drucker existieren die folgenden Befehlsschlüssel:

Drucken und Zeilentransport	0 LLLL 001
Sofortiger Zeilentransport	0 LLLL 011
Drucken und Vorschub	1 CCCC 001
Sofortiger Vorschub	1 CCCC 011

Hierbei können für LLLL die Bitfolgen 0000, 0001, 0010 und 0011 eingesetzt werden, wodurch ein Zeilentransport um 0, 1, 2 bzw. 3 Zeilen hervorgerufen wird. Für CCCC können die Bitfolgen 0001 bis 1100 eingesetzt werden, wodurch ein Vorschub bis zu einer Lochung in dem entsprechenden Kanal (1 bis 12) bewirkt wird. Neben diesen Befehlen gibt es noch den Befehl „Prüfen (Sense)", dessen Befehlsschlüssel 0 0000 100 ist.

Es wäre noch zu bemerken, daß ein sofortiger Zeilentransport um 0 Zeilen einem Nop-Befehl (keine Operation) gleichkommt. Das zweimalige Drucken einer Zeile macht das Druckbild kräftiger und wird manchmal dazu benutzt, um Überschriften aus dem übrigen Text etwas herauszuheben.

1) Im angloamerikanischen Sprachraum als „Forms Control Buffer", abgekürzt FCB, bezeichnet

2) 2,36 bzw. 3,15 Zeilen pro cm.

Das Gerätestatusbyte ist bei allen Druckern mit dem gleichen Inhalt versehen. Seinen Bitstellen kommt die nachstehende Bedeutung zu:

a) Bitstelle 0 — nicht benutzt

b) Bitstelle 1 — nicht benutzt

c) Bitstelle 2 — nicht benutzt

d) Bitstelle 3 — In Arbeit
Standard bei allen Druckern

e) Bitstelle 4 — Kanalende
Diese Bedingung wird bei allen Druckern dann signalisiert, wenn der Puffer gefüllt ist. Alle Drucker arbeiten mit einem Puffer.

f) Bitstelle 5 — Geräteende
Diese Bedingung tritt nach dem Abschluß des Druckvorganges auf

g) Bitstelle 6 — Gerätefehler
Diese Bitstelle enthält stets dann eine 1, wenn im Prüfbyte 0 ein Fehlerbit gesetzt ist.

h) Bitstelle 7 — Geräte-Ausnahmebedingung
Diese Bitstelle wird auf 1 gesetzt, wenn ein Transport über eine im Kanal 12 des Lochbandes vorkommende Lochung durchgeführt wird. Zur Aussendung kommt es zur Zeit der Feststellung der Geräteendebedingung.

Die Prüfbytes sind im wesentlichen bei allen Druckern gleich. Das Prüfbyte 0 enthält folgende Informationen:

a) Bitstelle 0 — Befehlszurückweisung
Undefinierter Befehlsschlüssel; bei einigen Druckern kommt es auch dann zur Zuweisung einer 1, wenn eine ungültige Befehlsfolge vorliegt.

b) Bitstelle 1 — Eingriff erforderlich
(1) Drücken der Stoptaste
(2) Die Verkleidung oder eine andere Sperre ist geöffnet
(3) Fehler in der Formularführung
(4) Fehler im Gleichlauf
(5) Papierende

c) Bitstelle 2 — Fehler in der Ausgangsleitung
Dieser Fehler wird zusammen mit Kanalende (Bitstelle 4 des Gerätestatusbytes) signalisiert

d) Bitstelle 3 — Ausrüstungsfehler
Bei Druckern führt der Versuch, die Operation ein zweites Mal zu wiederholen, gewöhnlich zum Erfolg

e) Bitstelle 4 — Datenfehler
Diese Stelle besitzt nur bei Druckern, die mit dem Universaltypensatz ausgerüstet sind, eine Bedeutung. Es wurde in den Druckpuffer ein Zeichen übertragen, das im Typensatz des Druckers nicht vorhanden ist. Zur Zuweisung einer 1 kommt es auch, wenn ein Vorschub zu einem Kanal versucht wird, der nicht im Formularsteuerungspuffer (FCB) definiert ist.

f) Bitstelle 5 — Paritätsprüfung
Fehler im UCS-Puffer oder im FCB

g) Bitstelle 6 —— Ladefehler
Während des Ladens des FCB wurden ungültige Daten entdeckt. Bei einigen Modellen wird außerdem dieser Bitstelle eine 1 zugewiesen, wenn nicht eine genügende Anzahl von Bytes zum UCS-Puffer gesendet werden.

h) Bitstelle 7 —— Kanal 9
Diese Bedingung entspricht der Geräte-Ausnahme-Bedingung, die für eine Lochung im Kanal 12 im Gerätestatusbyte registriert wird. Es wäre noch zu bemerken, daß ein über eine Lochung im Kanal 9 hinausgehender Zeilentransport gleichzeitig der Bitstelle 6 (Gerätefehler) des Gerätestatusbytes eine 1 zuweist.

Bei der IBM 1443 spezifizieren die Bitstellen 5 und 6 den montierten Druckstab (Typenstange) wie folgt: 00 für den 13-Zeichen-Vorrat, 01 für den 39-Zeichen-Vorrat, 10 für den 52-Zeichen-Vorrat und 11 für den 63-Zeichen-Vorrat.

Einige Drucker besitzen weitere Prüfbytes. Die anderen Bytes werden für die Fehlerbeseitigung und zur Wartung der Ausrüstung benötigt.

Wenn die Zusatzeinrichtung „Universaltypensatz" installiert ist, so ist damit gleichzeitig ein 240stelliger Zeichenpuffer eingebaut. Jede Stelle dieses Puffers muß dann mit einem Zeichen geladen werden, das dem an der gleichen Stelle des Typensatzes erscheinenden Zeichen entspricht. Hierfür werden Befehle verwendet, deren Befehlsschlüssel wir nicht beschrieben haben. Dieser Puffer muß selbstverständlich erneut geladen werden, wenn der Typensatz des Druckers, d.h. die Kette oder die Glieder, gewechselt wird. Wegen der zusätzlichen Mietkosten ist jedoch bei den meisten Installationen nur ein Typensatz vorhanden.

Bei Ausrüstung von Druckern mit dem Universaltypensatz gibt es zwei zusätzliche Befehle, die das Setzen der Bitstelle 4 des Prüfbytes 0 steuern. Es sind dies:

Setzen der Sperrung der Datenprüfung	(Set Data Check Block)	—— 73
Aufhebung der Sperrung der Datenprüfung	(Reset Data Check Block)	—— 7B

Der Befehl mit dem Befehlsschlüssel 7B (sed) bewirkt, daß durch die Bitstelle 4 des Prüfbytes die Bitstelle 6 des Gerätestatusbytes (Gerätefehler) auf 1 gesetzt wird, wenn ein Zeichen zum Drucker gesendet wird, das nicht im 240stelligen Puffer vorhanden ist. Der Befehl mit dem Befehlsschlüssel 73 (sed) unterdrückt dagegen das Setzen einer 1 in die Bitstelle 4. Ist der Universaltypensatz nicht eingebaut, so erzeugt das Drucken eines ungültigen Zeichens keine Fehleranzeige. Bei Installation dieser Zusatzeinrichtung steht dem Programmierer somit die Möglichkeit offen, zwischen einer Fehleranzeige und einer Unterdrückung derselben zu wählen. Wird die Fehleranzeige nicht gewünscht, so sollte der Befehl mit dem Schlüssel 73 ausgeführt werden.

Weitere Zusatzbefehle sind nachstehend beschrieben:

a) Unter „Falten" versteht man das Drucken von Kleinbuchstaben als Großbuchstaben. Hierfür gibt es die Befehle:

Falten (Fold)	—— 43
Entfalten (Unfold)	—— 23

b) Bei den Druckern des Typs IBM 1403 wird das Falten als Teil des Ladens des UCS-Puffers spezifiziert

Zulassung des UCS-Ladens (Allow UCS Load)	—— EB
Laden des UCS ohne Falten (Load UCS, no folding)	—— FB
Laden des UCS mit Falten (Load UCS, folding)	—— F3

Die beiden letzten Befehle müssen mit Befehlskettung an den ersten Befehl angehängt werden.

Bei der IBM 3211 wird der Befehl FB zum Laden des UCS-Puffers benutzt; eine Befehlskettung ist nicht erforderlich.

Die Drucker besitzen folgende Anzeigelampen:

a) NETZ EIN
Der Drucker ist mit Spannung versorgt.

b) BETRIEB
Der Drucker befindet sich im betriebsbereiten Zustand.

c) PAPIERENDE
Ungefähr 4 Zoll (10,16 cm) verbleiben noch zum Drucken, bevor das Papier zu Ende geht. Jedes Mal, wenn die Taste START gedrückt wird, kann in dieser Situation eine weitere Zeile gedruckt werden. Die Feststellung einer Lochung im Kanal 1 verhindert das weitere Drucken.

d) DRUCKPRÜFUNG
Ist der Drucker mit dem Universaltypensatz ausgerüstet, so wird diese Lampe eingeschaltet, wenn die Bitstelle 5 des Prüfbytes auf 1 gesetzt wird. Bei anderen Druckern leuchtet sie bei einem Defekt an den Hämmern oder bei einem Paritätsfehler auf. Der Bitstelle 3 des Prüfbytes wird eine 1 zugewiesen. Die Anzeigelampe verlöscht beim ersten nachfolgenden Befehl wieder, der kein Nop- oder Prüfbefehl ist.

e) GLEICHLAUFPRÜFUNG
Die Bewegung der Typen läuft nicht taktgleich mit dem Druckvorgang ab, die Synchronisation ist gestört. Diese Lampe verlöscht erst beim Drücken der Taste PRÜFUNG LÖSCHEN.

f) FORMULARFÜHRUNG
Das Papier wird nicht richtig zugeführt. Bei einigen Druckern leuchtet diese Lampe auch dann auf, wenn die Taste STOP gedrückt wird.

g) ABLAGE
Das Papier wird nicht mehr abgelegt, weil die Ablage gefüllt ist. Diese Bedingung wird zurückgesetzt, wenn der Drucker wieder arbeitsbereit ist.

h) FARBBAND
Zu dieser Fehleranzeige kommt es, wenn das Farbband nicht korrekt transportiert wird. Es ist ein manueller Eingriff erforderlich.

i) VAKUUM
Der Beutel des Vakuumreinigers ist voll, oder es hat sich übermäßig viel Staub und Schmutz angesammelt. Der Drucker kann weiterarbeiten, es sei denn, andere Bedingungen verhindern das.

j) VORSCHUB
Zu einem Vorschubfehler kommt es, wenn die Formulare zu weit weg oder nicht weit genug entfernt bewegt werden.

Nicht alle Typen besitzen die genannten Anzeigelampen. In gleicher Weise gibt es nicht alle der nachfolgend beschriebenen Tasten bei allen Druckertypen.

Die Arbeitsweise der Drucker kann durch die folgenden Tasten gesteuert werden:

a) NETZ EIN
Diese nur bei einigen Druckern vorhandene Taste verbindet bei ihrem Niederdrücken die Drucker mit dem Netz.

b) NETZ AUS
Das Drücken dieser nur bei einigen Druckern vorhandenen Taste trennt die Drucker vom Netz.

c) START oder DRUCK START oder DRUCKER START
Das Drücken dieser Taste versetzt den Drucker in den betriebsbereiten Zustand. Voraussetzung hierfür ist natürlich, daß der Drucker mit Spannung versorgt ist, der Typenträger an Ort und Stelle ist, die Sperren an der Maschine (die den Bediener vor Verletzung und Schäden schützen) geschlossen sind, das Papier eingespannt ist und keine Fehlerbedingung besteht.

d) STOP oder DRUCK STOP
Das Drücken dieser Taste löst den betriebsbereiten Zustand des Druckers ab. Bei einigen Druckern leuchtet gleichzeitig auch die Lampe FORMULARFÜHRUNG auf.

e) PRÜFUNG LÖSCHEN oder LÖSCHEN
Alle Fehleranzeigen werden beim Drücken dieser Taste gelöscht. Die Operationen können durch das Drücken der Taste START wieder aufgenommen werden.

f) VORSCHUB STOP
Das Drücken dieser Taste stoppt den Papiervorschub.

g) ZEILENTRANSPORT
Beim Drücken dieser Taste wird das Papier um eine Zeile weitertransportiert.

h) GRUNDSTELLUNG
Beim Drücken dieser Taste wird das Papier bis zu einer Lochung im Kanal 1 vorgeschoben.

i) EINZELZEILE
Durch Tastendruck wird der Drucker zum Drucken einer Zeile veranlaßt, danach ist er nicht betriebsbereit.

j) VORSCHUB STOP
Durch Tastendruck wird der manuelle Papiertransport ermöglicht. Damit wird auch unerwarteter Papierauslauf gestoppt. Diese Taste sollte nicht für das normale Anhalten des Druckers benutzt werden.

k) ABBRUCH
Durch Tastendruck wird die Bitstelle „Ausrüstungsfehler" auf 1 gesetzt. Die Taste ist nur im nicht betriebsbereiten Zustand funktionsfähig.

l) VERKL ZU
Das Drücken dieser nur bei einigen Druckern vorhandenen Taste hat zur Folge, daß sich die Verkleidung des Druckers schließt.

m) VERKL AUF
Das Drücken dieser ebenfalls nur bei einigen Druckern vorhandenen Taste bewirkt, daß sich die Verkleidung (Abdeckung) des Druckers öffnet.

Aufgabengruppe 13

1) Die Speicherstellen 002 000 bis 002 100 sollen auf jeder ihrer Bitstellen nur Nullen enthalten.

Es liegt weiterhin ein Kartenpaket vor, dessen Karten auf den Spalten 1 bis 5 Daten enthalten; die restlichen Spalten sind leer.

Welche Informationen sind auf den Speicherstellen 002 000 bis 002 003 zu der Zeit gespeichert, zu der es zu einer durch das Kanalprogramm

```
Ø2 ØØ2ØØØ 4Ø ØØØØ52
Ø2 ØØ2Ø52 ØØ ØØØØ5Ø
```

bewirkten Unterbrechung kommt? Es sollen der Reihe nach die Geräte

a) Kartenleser IBM 2501, b) Kartenleser IBM 2501 bei Dualkartenverarbeitung, c) Kartengerät IBM 2596 betrachtet werden!

Auf den ersten drei Karten sind folgende Daten gelocht:

(1) ABCDE
(2) 6/3=2
(3) $ **?.

2) Gib den Inhalt des CSW (Kanalstatuswortes) bei Aufgabe 1) an, wenn das CSW nur nach dem Auftreten der Geräteende-Bedingung gespeichert wurde.

3) Es soll die gleiche Aufgabenstellung wie bei Aufgabe 1) gelöst werden; das Kanalprogramm soll jedoch das folgende Aussehen haben:

```
22 ØØ2ØØØ 4Ø ØØØØ52
Ø2 ØØ2Ø52 ØØ ØØØØ5Ø
```

4) Es soll die gleiche Aufgabenstellung wie bei der Aufgabe 2) gelöst werden, nur diesmal unter Zugrundelegung des Kanalprogrammes von Aufgabe 3).

5) Wenn bei Aufgabe 2) das nächste für das betreffende Gerät bestimmte Kanalprogramm aus dem Befehl

```
Ø4 ØØ2Ø5Ø ØØ ØØØØØ1
```

bestehen würde, was würde dann im Speicherplatz mit der Adresse 002050 nach Ausführung dieses Programmes stehen?

6) Löse die gleiche Fragestellung wie bei Aufgabe 1) für das Kanalprogramm

```
22 ØØ2ØØØ 4Ø ØØØ1ØØ
C2 ØØ21ØØ 4Ø ØØØØ5Ø
E2 ØØ215Ø ØØ ØØØØ5Ø
```

Die Ausführung dieses Kanalprogrammes soll der Reihe nach für die folgenden Geräte untersucht werden:

a) IBM 2520 (Betriebsart 1)
b) IBM 2520 (Betriebsart 2), d.h. Dualkartenverarbeitung
c) IBM 3505 (Betriebsart 2), d.h. Dualkartenverarbeitung.

7) Es liegt die gleiche Aufgabenstellung wie bei Aufgabe 2) vor, jedoch unter Zugrundelegung des Programmes von Aufgabe 6).

8) Schreibe ein Überwachungsprogramm, das auf dem Speicherplatz 000 350 beginnt und das zur Prüfung der Konsolschreibmaschine IBM 3215 dienen kann. Wenn die Konsolschreibmaschine mit der Geräteadresse 01F verfügbar ist, so soll eine sieben Zeichen lange Nachricht herausgeschrieben werden, deren Anfangsadresse im Register 12 enthalten ist.

9) Auf drei aufeinanderfolgende Zeilen zu Beginn einer neuen Seite soll ein Name und die dazugehörige Adresse gedruckt werden. Die auf die einzelnen Zeilen zu druckenden

Daten sind auf den Speicherplätzen gespeichert, die mit den Adressen 002 000, 002 060 und 002 0C0 beginnen. Jede Zeile umfaßt gerade 96 Zeichen. Schreibe ein Kanalprogramm, mit dessen Hilfe die Zeilen gedruckt werden können. Nach dem Drucken soll der Drucker bis zu einer Lochung im Kanal 2 vorgeschoben werden; von dort aus sollen nämlich die Einzelzeilen gedruckt werden.

6.12. Magnetbänder

Magnetbänder werden hergestellt, indem man einen Überzug von Eisenoxyd auf einen Streifen plastischen Filmmaterials aufbringt. Das bei den Computern verwendete Magnetband unterscheidet sich von den Bändern, die bei den häuslichen Tonbandgeräten verwendet werden, nur in der Breite und in der Qualität. Die meisten Computerbänder sind $1\frac{1}{2}$ Zoll (3,81 cm) breit und 2400 Fuß (731,5 m) lang. Das Magnetband läuft unter Paaren von Lese/Schreibköpfen mit einer Geschwindigkeit zwischen 12,50 bis 200,00 Zoll/Sekunde (31,75 bis 508,00 cm/s) hindurch. Wenn neue Informationen auf das Magnetband aufgezeichnet werden sollen, so schreibt der erste Kopf das Binärmuster[1] auf das Band; der zweite Kopf liest anschließend die aufgezeichneten Informationen wieder. Stimmt das Gelesene nicht mit dem Geschriebenen überein, so wird ein Fehler angezeigt. Zum Lesen von zuvor aufgezeichneten Informationen passiert das Magnetband die gleichen Köpfe und wird von beiden Köpfen gelesen. Weichen die gelesenen Informationen voneinander ab, so wird ein Fehler angezeigt.

Die meisten Bandgeräte des Systems IBM /370 arbeiten gleichzeitig mit neun Paaren von Schreib/Lese-Köpfen; auf diese Weise sind auf dem Band neun Kanäle oder Spuren für die Aufzeichnung von Informationen verfügbar. Die zur gleichen Zeit aufgezeichneten neun Bits repräsentieren somit gerade ein Byte einschließlich der Bitstelle, die für Prüfzwecke (ungerade Anzahl von Einerbits) verwendet wird.

Drei Aufzeichnungsmethoden sind bei Magnetbändern gebräuchlich. Beim NRZ-Verfahren[2] wird auf das Band ein ständiger Impuls aufgezeichnet; ein Wechsel der Magnetisierungsrichtung zeigt hierbei das Bit 1, eine gleichbleibende Magnetisierungsrichtung das Bit 0 an. Das Phasenverschlüsselungsverfahren verursacht zwei Magnetflußänderungen bei einem Bit, das gleich dem vorhergehenden Bit ist, eine Flußänderung dagegen bei einem Bit, das sich von vorhergehenden unterscheidet. Das letztere Verfahren erfordert bei seiner Verwirklichung eine bessere Qualität der Magnetbänder und ist kostspieliger. Weist das Magnetband jedoch Schadstellen geringeren Umfanges auf, so können die meisten Fehler schneller und einfacher korrigiert werden. Bei der IBM /370 wird beim NRZ-Verfahren ein Magnetband mit einer Zeichendichte von 800 Bytes/Zoll (315 Bytes/cm) beschrieben, das Phasenverschlüsselungsverfahren verwendet eine Zeichendichte von 1600 Bytes/Zoll (630 Bytes/cm). Das Gruppenaufzeichnungsverfahren (Group Recording, abgekürzt GR) ähnelt dem Phasenverschlüsselungsverfahren (Phase Encoding, abgekürzt PE); es werden jedoch zusätzliche Bytes als Prüfbytes zwischen die Datenbytes gestellt, wenn der Satz aufgezeichnet wird. Beim Lesen werden diese Bytes wieder automatisch entfernt. Das GR-Verfahren arbeitet mit einer Zeichendichte von 6250 Bytes/Zoll (2445 Bytes/cm).

Bei einem Magnetband machen sich infolge des Abriebs der Oxydschicht auch Verschleißerscheinungen bemerkbar; es zeigen sich schlechte Stellen auf dem Band. Sind auf einem Magnetband nur wenige schlechte Stellen vorhanden, so ist es möglich, über sie hinwegzuspringen und den Rest des Bandes auszunutzen. Weist das Band jedoch schon zu viele

1) Auch Bitmuster genannt

2) NRZ ist die Abkürzung von Non Return to Zero, oft auch als Wechselschriftverfahren bezeichnet.

schadhafte Flecke auf, so sollte das Band aus dem Gebrauch gezogen werden. Mitunter werden Fehler auch durch Staubkörnchen hervorgerufen, die auf der Oberfläche des Bandes haften. Durch die physikalische Bewegung des Bandes werden diese Partikel oft einfach hinweggefegt, wodurch das Band wieder in einen einwandfreien Zustand gerät. Fehler dieser Art werden deshalb als vorübergehende Fehler bezeichnet; sie treten mehr oder minder bei allen Magnetbändern auf.

Magnetbänder kommen in verschiedenen Längen in den Handel, solche von 2400 Fuß (731,5 m) Länge sind jedoch die am meisten verwendeten. Da wir nicht eine ganze Rolle [1] durch eine einzige Operation beschreiben können bzw. auch wollen, ziehen wir das Aufzeichnen vieler kleiner Blöcke vor. Jeder Block ist hierbei von dem nachfolgenden Block durch eine 0,6 Zoll (1,54 cm) breite sogenannte Kluft[2] getrennt. Beim GR-Verfahren ist die Kluft jedoch nur 0,3 Zoll (0,76 cm) breit. Der Computer erkennt jede Kluft auf dem Band, die breiter als die angegebenen Werte ist, als eine Kluft an, die Satzblöcke voneinander trennt. Wenn wir auf eine schlechte Stelle des Magnetbandes stoßen, so überspringen wir einfach zusätzlich ein paar Zoll und versuchen erneut die Aufzeichnung von Informationen. Dadurch entsteht zwar eine breitere Kluft, doch hat diese keine weiteren Folgen. Es dauert nur etwas länger, die Schreib/Lese-Köpfe über diese Kluft hinwegzubewegen. Bei der IBM /370 kann die Blocklänge so groß gewählt werden, wie es die Kapazität des Hauptspeichers gerade noch erlaubt. Als minimale Blocklänge muß jedoch eine Länge von 18 Bytes eingehalten werden. Die meisten Programmierstandards diktieren jedoch als Maximum für die Blocklänge 32767 Bytes. Die durchschnittliche Blocklänge variiert, 4000 Bytes werden im Schnitt recht oft verwendet. Ein kleinerer Blockumfang als 18 Bytes kann als elektrischer Störimpuls angesehen werden. Eine Ausnahme macht jedoch die sogenannte Bandmarke. Darunter versteht man einen speziellen datenfreien Block. Besondere Schaltkreise im Bandgerät werden zur Feststellung von Bandmarken benutzt.

Auf der nicht für die Aufzeichnung vorgesehenen Seite des Magnetbandes ist in einer Entfernung zwischen 3 m und 4,50 m vom Anfang an ein ca. 0,48 cm (0,19 Zoll) langes reflektierendes Stück einer Aluminiumfolie befestigt. Dieser so markierte Punkt wird als Ladepunkt bezeichnet; durch ihn ist die Stelle des Magnetbandes festgelegt, von der aus die Aufzeichnung bzw. das Lesen von Informationen beginnen kann. Auf den wenigen Metern vom Anfang des Magnetbandes bis zum Ladepunkt können niemals Informationen aufgezeichnet werden. Ein gleiches Aluminiumstück markiert auch das Ende des Magnetbandes [3]. Diese Endmarke sollte jedoch so weit entfernt vom physikalischen Ende des Bandes angebracht werden, daß einige Blöcke auch noch nach Feststellung dieser Marke auf das Band aufgezeichnet werden können. Im allgemeinen vervollständigt man bei Verwendung eines Magnetbandes den letzten aufzuzeichnenden Block vor dem Hinausschreiben und schreibt hinter ihm noch einen speziellen Block auf das Band, der als Indikator, daß keine weiteren Daten auf dem Band stehen, dienen soll. Zu diesem Zweck wird die sogenannte Bandmarke [4] benutzt. Die Bandmarke stellt nichts anderes als einen Block mit spezieller Verschlüsselung dar, durch die ihn die Magnetbandgeräte von Datenblöcken unterscheiden können.

---

1) Eine ganze Rolle von 731,5 m Länge faßt bei einer Zeichendichte von 315 Bytes/cm insgesamt 23 000 000 Bytes, wenn keine Klüfte vorhanden sind.

2) Im Englischen: Inter-Block-Gap (abgekürzt IBG)

3) Im angloamerikanischen Sprachraum als „End of Tape", abgekürzt EOT, bezeichnet.

4) Im angloamerikanischen Sprachraum als „Tape Mark", abgekürzt TM, bezeichnet.

Die Richtigkeit der aufgezeichneten Daten wird drei verschiedenen Prüfungen unterworfen. Die Longitudinal- oder Längsprüfung [1] besteht darin, daß jedes aufgezeichnete Byte unter Einschluß des Paritätsbits auf eine ungerade Anzahl von Einerbits geprüft wird. Die Vertikal- oder Querprüfung [2] besteht darin, daß ein Extrazeichen an das Ende eines jeden Blockes geschrieben wird. Dieses Extrazeichen wird so gebildet, daß jede Spur gerade eine ungerade Anzahl von Einerbits enthält. Beim Lesen des Bandes wird dieses Prüfzeichen erneut gebildet und mit dem Zeichen verglichen, das auf dem Magnetband aufgezeichnet ist; es wird niemals zum Kanal gesendet. Die dritte Prüfung ist die Zyklusprüfung [3]. Diese Prüfung schließt ebenfalls ein Extrazeichen ein, das zu der Zeit bestimmt wird, zu der der Block geschrieben wird. Das von der Zyklusprüfung gebildete Zeichen wird unmittelbar vor dem Zeichen aufgezeichnet, das durch die Querprüfung generiert wird. Die Hauptaufgabe des Zykluszeichens besteht darin, die Wiederherstellung von Blöcken zu ermöglichen, bei denen nur Einbitfehler aufgetreten sind, d.h. der Block wurde bis auf ein Bit fehlerfrei gelesen. Wenn ein Lesefehler auftritt, wird das CRC-Byte zur Errechnung eines Spurfehlerbytes [4] benutzt. Durch Übertragen des TIE-Bytes zur Bandsteuereinheit, Zurücksetzen des Bandes und Neulesen des Blockes verursacht die Steuereinheit, daß die Paritätsspur benutzt wird, um den korrekten Inhalt der schadhaften Spur wiederherzustellen. Das TIE-Vorgehen versagt, wenn mehr als eine Spur defekt ist. Beim Lesen werden die Prüfbytes nicht in den Speicher übertragen.

Beim Phasenverschlüsselungsverfahren wirft das Lesen eines Bandes mit einem Einbitfehler kein Problem auf. Der Fehler wird sofort erfaßt, und die Bandsteuereinheit korrigiert ihn, bevor das betreffende Byte in den Hauptspeicher übertragen wird. Dieses Verfahren verwendet deshalb nicht das Zyklusprüfzeichen. Zusätzlich werden am Ende jeden Blockes vierzig Synchronisierungszeichen aufgezeichnet. Diese Bytes dienen dem Gerät zur Ausrichtung des Bandes. Die Natur des Phasenverschlüsselungsverfahrens [5] erlaubt die automatische Entdeckung und Korrektur aller Bytes, bei denen ein Bit fehlerhaft ist. Somit wird beim PE-Verfahren ein korrigiertes Byte an den Hauptspeicher gesendet, ohne daß eine spezielle Programmierung für die Fehlerbeseitigung erfolgen muß.

Das GR-Verfahren verhält sich ähnlich dem PE-Verfahren. Die Gruppenprüfbytes helfen während des Bandlesens bei der automatischen Beseitigung von Zwei-Bit-Fehlern.

Bei allen Aufzeichnungsverfahren werden Zwei-Bit-Fehler entdeckt. In einigen Fällen erfolgt keine automatische Korrektur. Die normale Fehlerbeseitigungsprozedur besteht darin, den Block viele Male neu zu lesen. Bei Fortbestehen des Fehlers sind die Daten effektiv verlustig gegangen.

Um dem zufälligen Beschreiben einer Rolle, auf der wertvolle Informationen aufgezeichnet sind, vorzubeugen, wird ein Schreibring (Dateischutzring) verwendet. Der in Abb. 6.32. dargestellte Ring paßt gerade in die kreisförmige Vertiefung der Bandrolle. Wenn der Ring nicht eingesetzt ist, kann das Bandlaufwerk keine Schreiboperationen ausführen; somit sind die auf dem Band stehenden Informationen gegen das Überschreiben geschützt.

---

1) Im angloamerikanischen Sprachraum als „Longitudinal Redundancy Check", abgekürzt LRC, bezeichnet
2) Im angloamerikanischen Sprachraum als „Vertical Redundancy Check", abgekürzt VRC, bezeichnet
3) Im angloamerikanischen Sprachraum als „Cyclic Redundancy Check", abgekürzt CRC, bezeichnet
4) Im angloamerikanischen Sprachraum als „Track in Error"-Byte, abgekürzt TIE-Byte, bezeichnet
5) Im angloamerikanischen Sprachraum als „Phase Encoding", abgekürzt PE, bezeichnet

Abb. 6.32. Magnetbandrolle und Schreibring (Dateischutzring)

6.13. Das Magnetbandgerät IBM 3420

In Abb. 6.33. ist das Magnetbandgerät IBM 3420 abgebildet. Zwischen 1 und 16 solcher Geräte können an eine Bandsteuereinheit[1] IBM 3803 angeschlossen werden. Sondereinrichtungen ermöglichen, daß jede IBM 3420 mit 1, 2, 3 oder 4 Bandsteuereinheiten verbunden werden kann. Jede Steuereinheit wiederum kann an 1 oder 2 Kanäle angeschlossen werden, die zur gleichen Zentraleinheit oder zu verschiedenen Zentraleinheiten gehören können. In den meisten Fällen sind vier bis acht IBM 3420 an eine IBM 3803 angeschlossen, die stets abwechselnd mit einem anderen Selektorkanal verbunden ist.

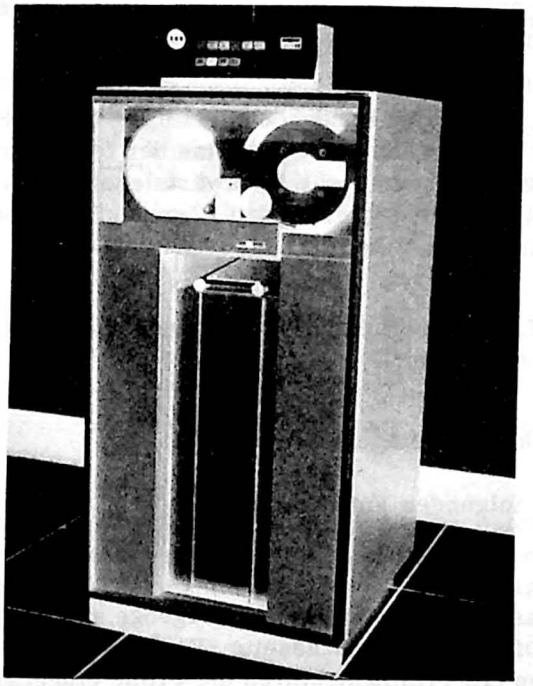


Abb. 6.33. Das Magnetbandgerät IBM 3420

1) Im angloamerikanischen Sprachraum als „Tape Control Unit", abgekürzt TCU, bezeichnet.

Die Bedienungseinrichtungen der IBM 3420 sind in Abb. 6.34. dargestellt.

Abb. 6.34. Die Bedienungs- und Kontrolleinrichtungen des Magnetbandgerätes IBM 3420

Die sechs Anzeigelampen haben folgende Bedeutungen:

a) AUSWAHL
Diese Lampe leuchtet auf, wenn das betreffende Gerät adressiert ist, d.h. eine Operation ausführt.

b) BEREIT
Diese Lampe leuchtet auf, wenn das Magnetband in die Vakuumkanäle (Vakuumschächte) eingezogen, die zum Einsetzen der Rollen vorhandene Tür bzw. das Fenster geschlossen und die Taste START gedrückt ist. Außerdem darf das Bandgerät kein Rückspulen durchführen.

c) SCHREIBSPERRE
Diese Lampe leuchtet auf, wenn der Schreibring nicht auf die Rolle aufgezogen ist. Die Lampe leuchtet aber auch auf, wenn keine Rolle in das Bandgerät eingesetzt ist oder wenn eine Rückspuloperation im Gange ist.

d) BANDANZEIGE
Diese Lampe leuchtet auf, wenn eine Bandendemarkierung (EOT) entdeckt wurde. Diese Lampe verlöscht jedoch wieder, wenn das Gerät eine der folgenden Operationen ausführt: Rückspulen, Rückspulen und Entladen sowie Rückwärtslesen. Sie verlöscht aber ebenfalls, wenn die Taste ENTLD. RÜCKSP. gedrückt wird. Zum Aufleuchten kommt es aber auch, wenn das Magnetband gerissen ist.

e) NETZFEHLER
Beim Aufleuchten dieser Lampe liegt ein Hardwareproblem vor, das nur durch das Wartungspersonal behoben werden kann.

f) LADEFEHLER
Das Magnetband ist nicht korrekt eingefädelt.

Die vier Tasten erfüllen die folgenden Funktionen:

a) LADEN
RÜCKSP.
Nach Einlegen der Bandkassette laufen folgende Vorgänge ab: Schließen der Fenster, Verriegelung der Kassette, Öffnen der Bandkassette, Einfädelung des Magnetbandes (in fast der gleichen Weise werden bei Filmprojektoren die Filme eingefädelt), Einziehen des Bandes in die Vakuumschächte. Das geladene Band wird anschließend bis zum Ladepunkt zurückgespult und angehalten. Die Taste funktioniert nicht, wenn die Anzeigelampe BEREIT aufleuchtet.

b) START
Durch das Drücken dieser Taste wird das Bandgerät in den betriebsbereiten Zustand versetzt und die Lampe BEREIT zum Aufleuchten gebracht. Voraussetzung hierfür ist, daß alle Bedingungen für den betriebsbereiten Zustand erfüllt sind.

c) ENTLAD.
RÜCKSP.
Das Magnetband wird bis zum Ladepunkt zurückgespult und aus den Vakuumschächten herausgezogen. Danach wird die Kassette geschlossen und entriegelt und das Bedienerfenster geöffnet, so daß die Bandkassette nun entnommen werden kann. Die Taste ist bei Aufleuchten der Lampe BEREIT unwirksam.

d) STEUER
LÖSCH.
Wenn die Lampe LADEFEHLER aufleuchtet, wird sie durch Drücken dieser Taste zum Verlöschen gebracht. Ist das Bandgerät betriebsbereit, wird die Lampe BEREIT ausgeschaltet, es sei denn, daß eine Rückspuloperation im Gange ist. Erfolgt das Drücken während des Hochgeschwindigkeitsrückspulens, wird zum langsameren Rückspulen umgewechselt. Drücken beim langsamen Rückspulen führt zum Aufleuchten des Bandes.

Für Magnetbänder werden oft Spezialkassetten benutzt. Diese haben die Eigenschaft, daß die IBM 3420 sie öffnen und nach Verarbeitung des Bandes auch wieder schließen kann, wobei der Datenträger nicht vom Bediener berührt zu werden braucht. Um eine Kassette zu montieren, braucht man nur die Kassette auf die Nabe zu setzen und die Tasten LADEN RÜCKSP. und START zu drücken. Um ein Band zu montieren, das sich nicht in einer solchen Spezialkassette befindet, muß man die Bandrolle aus dem Behälter nehmen und sie zunächst auf die Nabe setzen. Danach ist die Rolle solange zu drehen, bis das Bandende in die Einfädelungsbahn fällt, die unmittelbar unter der Nabe liegt. Zum Schluß sind wieder die Tasten LADEN RÜCKSP. und START zu drücken.

Um ein Band zu entladen, sind einfach die Tasten STEUER. LÖSCH. und ENTLAD. RÜCKSP. zu drücken. Nach Öffnen des Fensters ist die Kassette bzw. die Bandrolle zu entnehmen.

Ein verstümmeltes Band kann nicht so einfach geladen werden. In diesem Fall muß das Bandende neu zugeschnitten werden, und zwar so weit zurückgesetzt, daß das Band völlig glatt vorliegt.

Es gibt folgende Befehlsschlüssel für die IBM 3420:

a) 02 — Lesen vorwärts (Read forward)
Das Band wird bis zur nächsten Kluft weiterbewegt. Die Daten werden zum Kanal übertragen.

b) 0C — Lesen rückwärts (Read backwards)
Das Band wird in entgegengesetzter Richtung bis zur nächsten Kluft weiterbewegt. Die gelesenen Daten werden zum Kanal übertragen. Im Hauptspeicher werden die Daten in absteigende Speicherstellen gespeichert, beginnend mit dem Speicherplatz, dessen Adresse im CCW angegeben ist.

c) 04 — Prüfen (Sense)
Prüfinformationen von 24 Bytes werden zum Kanal übertragen. Die Prüfbytes werden zu Beginn jeder Lese- bzw. Schreiboperation gelöscht, aber auch am Anfang jeder Steueroperation, die eine Bewegung des Magnetbandes erforderlich macht.

e) 07 — Rückspulen (Rewind-REW)
Durch diese Operation wird das Band zurückgespult und anschließend auf den Ladepunkt eingestellt. In dem Augenblick, in dem mit dem Rückspulen begonnen wird, wird die Bedingung „Geräte" signalisiert. Es findet keine Datenübertragung statt. Diese Operation wird vom Gerät selbst ohne Beteiligung der Steuereinheit ausgeführt. Die Bedingung „Geräteende" wird dann zm zweiten Male signalisiert, wenn das Rückspulen beendet ist.

f) 0F — Rückspulen und Entladen (Rewind and Unload-RUN)
Diese Operation hat das Entladen des Magnetbandes zur Folge; ansonsten liegen die gleichen Verhältnisse wie bei der Operation „Rückspulen" vor.

g) 17 — Löschen Kluft (Erase Gap-ERG)
Bei dieser Operation handelt es sich um eine Schreiboperation ohne Daten. Auf etwa 4,2 Zoll (10,7 cm) Länge werden alle Aufzeichnungen gelöscht, wodurch eine vergrößerte Kluft entsteht. Wenn der Befehl mehrere Male aufeinanderfolgend gegeben wird, so löscht nur der erste die erwähnten 4,2 Zoll Bandlänge, bei den nachfolgenden sind es nur je 3,6 Zoll (9,1 cm). Durch diesen Befehl kann über schadhafte Bandstücke hinweggegangen werden, falls das beim Beschreiben des Bandes erforderlich sein sollte.

h) 97 — Löschen mit Datenschutz (Data Security Erase-DSE)
Dieser Befehl wird nur dann ausgeführt, wenn er an einen Befehl „Löschen Kluft" gekettet ist. Das Band wird bis zur Bandendemarkierung (EOT) gelöscht. Daten, die hinter dieser aufgezeichnet sind, bleiben erhalten.

i) 1F — Schreiben Bandmarke (Write Tape Mark-WTM)
Durch diesen Befehl wird ein spezieller kurzer Block geschrieben, der als Bandmarke bezeichnet wird. Wie bei jedem anderen Block steht auch vor und nach diesem Block eine Kluft, die nachfolgende Kluft ist länger als die sonstigen Klüfte.

j) 27 — Zurücksetzen um eine Blocklänge (Backspace Block-BSB)
Das Magnetband wird bis zur nächsten Kluft rückwärts bewegt. Dabei werden die aufgezeichneten Informationen vom Gerät zur Steuereinheit übertragen, jedoch nicht bis zum Kanal. Somit ist der Kanal frei, um einen anderen Befehl ausführen zu können; die Steuereinheit ist jedoch solange beschäftigt, bis die Kluft erreicht ist.

k) 2F — Zurücksetzen um eine Datei (Backspace File-BSF)
Das Magnetband wird bis hinter die nächste Bandmarke rückwärts bewegt. Wenn keine E marke gefunden wird, bleibt das Band am Ladepunkt stehen. Das Band wird so eingestell daß ein nachfolgender Lesebefehl zum Lesen der Bandmarke führt. Der Kanal wird in de Augenblick frei zur Ausführung eines anderen Befehls, in dem die Operation beginnt. Di Steuereinheit bleibt jedoch bis zum Ende der Bandbewegung beschäftigt.

l) 37 — Übergehen eines Blockes (Forward Space Block-FSB)
Es läuft der gleiche Vorgang wie beim Befehl BSB ab, nur daß das Band vorwärts beweg wird.

m) 3F — Übergehen einer Datei (Forward Space Block-FSF)
Es läuft der gleiche Vorgang wie beim Befehl BSF ab, nur daß das Band vorwärts beweg wird.

Es gibt fünf Steuerbefehle, durch die keine Bandbewegung hervorgerufen wird:

a) C3 — Einstellen auf die Betriebsart 1600 PE (Set 1600 PE Mode)
Bei Magnetbandgeräten, die so ausgerüstet sind, daß sie die Aufzeichnung mit einer Zeichendichte von 1600 Zeichen/Zoll im Phasenverschlüsselungsverfahren durchführen, bedeutet dieser Befehl gewissermaßen eine Löschbedingung, d.h. das Bandgerät

kehrt wieder zu seiner normalen Aufzeichnungstechnik zurück und verläßt die Aufzeichnungstechnik des NRZ-Verfahrens.

b) CB —— Einstellen auf die Betriebsart 800 NRZ (Set 800 NRZ Mode)
Dieser Befehl kann nur für Bandgeräte gegeben werden, die mit den Zeichendichten 1600 und 800 Zeichen/Zoll arbeiten können. Die Befehle mit den Befehlsschlüsseln C3 und CB werden von der Steuereinheit ausgeführt, berühren also das EA-Gerät selbst nicht. Sie haben zur Folge, daß die Steuereinheit die angegebene Zeichendichte auf alle Bänder anwendet, die hinterher auf den Ladepunkt eingestellt werden. Die Angabe der Zeichendichte bezieht sich, wie schon gesagt, selbstverständlich nur auf die Schreiboperationen. Wenn sich ein Magnetband nicht im Ladepunkt befindet, so haben die Befehle zur Einstellung auf eine bestimmte Betriebsart erst dann einen Effekt, wenn das Band anschließend zurückgespult wird.

c) D3 —— Einstellen auf die Betriebsart 6250 GR (Set 6250 GR Mode)
Dieser Befehl kann nur für Bandgeräte gegeben werden, die mit einer Zeichendichte von 6250 Zeichen/Zoll operieren können.

Beim Lesen wird das Bandlaufwerk automatisch auf die Zeichendichte eingestellt, die das montierte Band besitzt. Die Moduseinstellbefehle werden von der Bandsteuereinheit ausgeführt. Sie haben zur Folge, daß die Steuereinheit auf dieser Zeichendichte eingestellt bleibt; das gilt für alle nachfolgenden Bänder bis zum Ladepunkt. Befindet sich das Band nicht im Ladepunkt, wird der Befehl erst dann wirksam, wenn das Band den Ladepunkt erreicht. Die Zeichendichtefestlegung wird nur auf Schreibbefehle bezogen.

d) DB —— Anforderung des Spurfehlerbytes (Request TIE)
Dieser Befehl bewirkt, daß das Spurfehlerbyte (TIE-Byte) zur Steuereinheit übertragen wird. Dieser Befehl hat nur für Bandgeräte mit einer Zeichendichte von 800 Zeichen/Zoll Gültigkeit.

e) F4 —— Reservieren (Reserve)
Dieser Befehl bewirkt, daß die Bandsteuereinheit für den Kanal reserviert wird, von dem der Befehl empfangen wurde. Die Steuereinheit antwortet mit „Steuereinheit in Arbeit", wenn Befehle von anderen Kanälen ankommen.

Die Bitstellen des Gerätestatusbytes haben folgende Bedeutung:

a) Die Bitstelle 0 hat keine Bedeutung.

b) Bitstelle 1 —— Statusänderung
Tritt auf dieser Bitstelle eine 1 zusammen mit einer 1 in der Bitstelle 3 (in Arbeit) auf, so bedeutet das, daß die Steuereinheit beschäftigt ist.

c) Bitstelle 2 —— Steuereinheitenende
Diese Bitstelle wird in den folgenden beiden Fällen auf 1 gesetzt:
(1) Nach Beendigung jeder Operation, während der die Steuereinheit beschäftigt war.
(2) Nach Beendigung jeder Steueroperation, während der ein Gerätefehler oder eine Geräte-Ausnahmebedingung entdeckt wurde.

d) Bitstelle 3 —— In Arbeit
Enthält zusammen mit dieser Bitstelle auch die Bitstelle 1 eine 1, so ist die Steuereinheit beschäftigt; enthält nur diese Bitstelle eine 1, so ist das EA-Gerät in Arbeit.

e) Bitstelle 4 —— Kanalende
Bei Steuerbefehlen wird diese Bedingung sofort signalisiert, bei Lese- bzw. Schreibbefehlen erst nach Beendigung der Datenübertragung.

f) Bitstelle 5 — Geräteende
Eine 1 auf dieser Bitstelle zeigt an, daß das Gerät seine Operation beendet hat. Geräteende wird auch dann signalisiert, wenn ein Gerät vom nicht betriebsbereiten Zustand in den betriebsbereiten übergeht.

g) Bitstelle 6 — Gerätefehler
Dieser Bitstelle wird eine 1 in den folgenden Fällen zugewiesen:
(1) Irgendeine Bitstelle des Prüfbytes 0 enthält eine 1.
(2) Bei einer Rückwärtsbewegung des Magnetbandes wird der Ladepunkt erreicht.
(3) Die Steuereinheit hat ihren Anteil beim Befehl „Rückspulen und Entladen" geleistet.
(4) Bitstelle 7 des Prüfbytes 1 ist auf 1 gesetzt.
(5) Bitstelle 3 des Prüfbytes 5 ist auf 1 gesetzt.
(6) Das Gerät ist am Ende der Befehle „Rückspulen" und „Löschen mit Datenschutz" nicht arbeitsbereit.
(7) Es tritt ein Ladefehler auf.

h) Bitstelle 7 — Geräte-Ausnahmebedingung
Dieser Stelle wird die Ziffer 1 immer dann zugewiesen, wenn bei einem der Befehle Schreiben (Befehlsschlüssel 01), Schreiben Bandmarke (Schlüssel 1F) oder Löschen Kluft (Schlüssel 17) das Reflexionsblättchen, das das Bandende (EOT) markiert, passiert wird. Die gleiche Zuweisung erfolgt, wenn während der Ausführung von einem der Befehle Lesen (02), Lesen rückwärts (0C), Übergehen eines Blockes (37), Zurücksetzen um eine Blocklänge (27) eine Bandmarke entdeckt wird. Eine Geräte-Ausnahmebedingung wird dagegen nicht bei den Befehlen BSF und FSF signalisiert.

Die wichtigsten Prüfbitstellen der Prüfbytes haben folgende Bedeutungen:

1. Prüfbyte 0

a) Bitstelle 0 — Befehlszurückweisung
Dieser Bitstelle wird eine 1 bei ungültigen Befehlsschlüsseln oder bei einem der Befehle Schreiben (01), WTM (1F) bzw. ERG (17) zugewiesen; bei den angeführten drei Befehlen ist das jedoch nur dann der Fall, wenn diese auf eine Datei gerichtet sind, die auf einem geschützten Magnetband residiert, d.h. einem Magnetband ohne Schreibring. Außerdem werden Befehle zurückgewiesen, wenn Reservierung und Freigabe auf eine Steuereinheit gerichtet sind, die nur mit einem anderen Kanal verbunden werden kann. Reservierung und Freigabe, die an einen anderen Befehl gekettet sind, sind ebenfalls nicht erlaubt.

b) Bitstelle 1 — Eingriff erforderlich
Die Bitstelle 1 des Prüfbytes 1 enthält eine 0. Wird immer dann gesetzt, wenn ein Gerät nicht betriebsbereit ist.

c) Bitstelle 2 — Fehler in der Ausgangsleitung

d) Bitstelle 3 — Ausrüstungsfehler

e) Bitstelle 4 — Datenprüfung
Daten wurden nicht korrekt geschrieben oder gelesen. Dazu kommt es immer dann, wenn der Versuch unternommen wird, ein Band zu lesen, das mit einer Aufzeichnungsdichte beschrieben ist, zu deren Lesen das Gerät nicht in der Lage ist.

f) Bitstelle 5 — Datenverlust
Dieser Bitstelle kann während des Vorwärts- und Rückwärtslesens sowie beim Schreiben eine 1 zugewiesen werden, wenn Daten verlorengehen. Jeder Datenverlust stoppt die weitere Übertragung.

g) Bitstelle 6 —— Wortzähler Null
Diese Bedingung wird gesetzt, wenn ein Schreibbefehl endet, bevor die Bandbewegung startet. Kann auch durch eine HIO-Instruktion gesetzt werden, wenn irgendein Befehl ohne Bandbewegung endet.

h) Die Bitstelle 7 hat keine Bedeutung.

2. Prüfbyte 1

a) Bitstelle 0 —— Störung
Beim NRZ-Verfahren wird dieser Bitstelle immer dann eine 1 zugewiesen, wenn fremde Daten in einer Kluft entdeckt werden. Beim Phasenverschlüsselungsverfahren enthält diese Bitstelle stets die Ziffer 0.

b) Bitstelle 1 —— Bandeinheit betriebsbereit
Diese Bitstelle enthält immer dann eine 1, wenn das betreffende Bandgerät betriebsbereit ist.

c) Bitstelle 2
Enthält diese Bitstelle eine 1 und ist gleichzeitig das Bit 1 dieses Bytes gleich 0, so existiert das betreffende Bandgerät nicht. Enthalten diese Bitstelle und die Bitstelle 1 dieses Bytes gleichzeitig eine 1 und ist das Bit 3 (In Arbeit) des Gerätestatusbytes gleich 0, so kann das Magnetbandgerät nicht zu einer Operation herangezogen werden; ist dagegen unter sonst gleichen Bedingungen der Bitstelle 3 des Gerätestatusbytes eine 1 zugewiesen, so ist das Bandgerät mit einer Rückspuloperation beschäftigt oder einer anderen Steuereinheit zugewiesen.

d) Bitstelle 3
Ohne Bedeutung.

e) Bitstelle 4
Das Magnetband ist auf den Ladepunkt eingestellt.

f) Bitstelle 5
Das Band befindet sich im Schreibzustand. Das bedeutet, daß die letzte Bandbewegung durch einen der folgenden Befehle bedingt war: Schreiben (01), WTM (1F), ERG (17).

g) Bitstelle 6
Das Band ist gegen das Beschreiben geschützt, d.h. der Schreibring ist nicht aufgezogen.

h) Bitstelle 7
Das Bandgerät ist nicht in der Lage, in der Betriebsart zu schreiben, die von den Einstellbefehlen (Befehlsschlüssel C3, CB, D3) verlangt wird.

3. Prüfbyte 2

Das Prüfbyte 2 enthält beim NRZ-Verfahren das Spurfehlerbyte (TIE-Byte). Weisen dabei die beiden Bitstellen 6 und 7 gleichzeitig eine 1 auf, dann liegt entweder kein Fehler vor oder mehr als eine Spur weist Fehler auf.

4. Prüfbyte 3

a) Bitstelle 5
Das Bandgerät ist auf eine Zeichendichte von 1600 Zeichen/Zoll gesetzt.

b) Bitstelle 6
Die letzte Bandbewegung war rückwärts.

Typ und Modell	Zeichendichte (Zeichen/Zoll)	Bandsteuergerät	Anzahl der Bandlaufwerke	Bandgeschwindigkeit in Zoll/s
2415-1	800	L oder S	2	18,75
2	800	L oder S	4	18,75
3	800	L oder S	6	18,75
4	1600;800	L oder S	2	18,75
5	1600;800	L oder S	4	18,75
6	1600;800	L oder S	6	18,75
2401-1	800	-	1	37,50
2	800	-	1	75,00
3	800	-	1	112,50
4	1600;800	-	1	37,50
5	1600;800	-	1	75,00
6	1600;800	-	1	112,50
2402-1	800	-	2	37,50
2	800	-	2	75,00
3	800	-	2	112,50
4	1600;800	-	2	37,50
5	1600;800	-	2	75,00
6	1600;800	-	2	112,50
2403-1	800	L oder S	1	37,50
2	800	L oder S	1	75,00
3	800	L oder S	1	112,50
4	1600;800	L oder S	1	37,50
5	1600;800	L oder S	1	75,00
6	1600;800	L oder S	1	112,50
2404-1	800	L und S	1	37,50
2	800	L und S	1	75,00
3	800	L und S	1	112,50
2420-5	1600;800	-	1	100,00
7	1600;800	-	1	200,00
2803-1	**800**	**L oder S**	**Steuereinheiten**	
2	**1600;800**	**L oder S**		
2804-1	**800**	**L und S**	**Steuereinheiten**	
2	**1600;800**	**L und S**		
3410-1	1600	-	1	12,50
2	1600;800	-	1	25,00
3	1600;800	-	1	50,00
3411-1	1600	L oder S	1	12,50
2	1600;800	L oder S	1	25,00
3	1600;800	L oder S	1	50,00
3420-3	1600;800	-	1	75,00
4	6250;1600	-	1	75,00
5	1600;800	-	1	125,00
6	6250;1600	-	1	125,00
7	1600;800	-	1	200,00
8	6250;1600	-	1	200,00
3803-1	**1600;800**	**L oder S**	**Steuereinheiten**	
2	**6250;1600;800**	**L oder S**		

Zeichenerklärungen: L Lesen
S Schreiben

Abb. 6.35. Übersicht über Magnetbandgeräte

5. Prüfbyte 4

Die Bitstelle 2 enthält eine 1, wenn die Bandanzeige eingeschaltet ist.

6. Prüfbyte 6

Die Bitstellen 4 bis 7 enthalten einen Code für die Modellnummer.

0011	Modell 3
0100	Modell 5
0101	Modell 7

6.14. Andere Magnetbandgeräte

In der Abb. 6.35. sind die Charakteristika von Magnetbandgeräten und Steuereinheiten aufgeführt, die an das System IBM /370 angeschlossen werden können. Ist mehr als eine Aufzeichnungsdichte angegeben, so gilt die zuerst angegebene als Standarddichte, die anderen sind als Sonderausrüstung „verschiedene Schreibdichte" verfügbar. Eine Steuereinheit, für die „L oder S" notiert ist, gilt als normale Einheit, so wie wir sie vom letzten Abschnitt her kennen. Bei einer Einheit mit dem Vermerk „L und S" handelt es sich um eine mit zwei Kanälen verbundene Steuereinheit, die fähig ist, gleichzeitig eine Lese- und eine Schreiboperation ausführen zu können. Sie ist keinesfalls äquivalent zwei getrennten Steuereinheiten, da sie nicht zwei Lese- oder zwei Schreiboperationen zur gleichen Zeit erlaubt.

Bandkassetten und automatisches Einfädeln sind nur für die IBM 2420 und die IBM 3420 verfügbar. Bei allen anderen Geräten muß der Maschinenbediener manuell den Anfang des Magnetbandes auf die Aufnahmerolle (Maschinenrolle) wickeln. Die beiden Abb. 6.36. und 6.37. vermitteln einen recht guten Eindruck über den dadurch zu treibenden Aufwand.

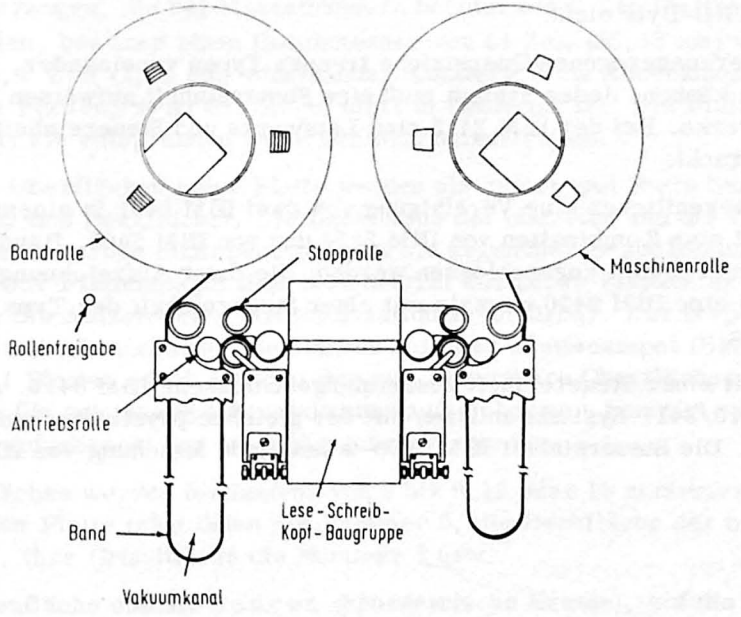


Abb. 6.36. Die Bandführung bei den Magnetbandgeräten des Typs IBM 2401

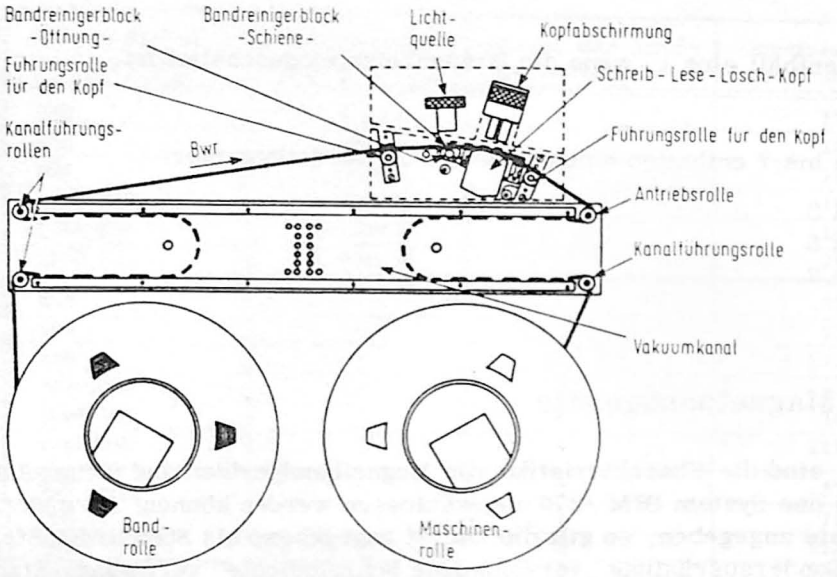


Abb. 6.37. Die Bandführung bei den Magnetbandgeräten des Typs IBM 2415

Motorgetriebene Fensteröffner gibt es bei den Geräten der Typen IBM 2415, IBM 3410, IBM 3411 und bei einigen ersten Produktionsserien der Typen IBM 2401, IBM 2402 und IBM 2403 nicht. Bei diesen Typen muß der Maschinenbediener Fenster oder Tür mit der Hand selbst öffnen.

Geräte mit Typennummern unter 3000 benutzen sechs Prüfbytes, die IBM 3410 und IBM 3411 sind auf neun abgestellt, die IBM 3420 auf 24. Die ersten drei Prüfbytes sind identisch bei allen Geräten. Die Befehle gleichen sich im wesentlichen, den DSE-Befehl gibt es allerdings nur bei den Geräten vom Typ IBM 3420. Geräte vom Typ IBM 2415 prüfen beim Lesen das CRC-Byte nicht.

Die dicken, nach links herausgezogenen Querstriche trennen Typen voneinander, die nicht gemischt benutzt werden können. Jedes System muß eine Steuereinheit aufweisen und ein oder mehrere Bandlaufwerke. Bei der IBM 2415 sind Laufwerke und Steuereinheiten in einem Gehäuse untergebracht.

Die IBM 2402 stellt im wesentlichen eine Vereinigung von zwei IBM 2401 in einem Gehäuse dar. Die IBM 2403 ist eine Kombination von IBM 2804 und von IBM 2401. Bandgeräte können nur über eine Steuereinheit angeschlossen werden, die ihrer Aufzeichnungsdichte angepaßt ist. Somit kann eine IBM 2420 niemals mit einer Steuereinheit des Typs IBM 2803-1 verbunden werden.

Die IBM 3411 ist eine mit einer Steuereinheit zusammengeschlossene IBM 3410. Alle Laufwerke eines IBM 3410/3411-Systems müssen mit der gleichen physikalischen Geschwindigkeit operieren. Die Steuereinheit IBM 3803-2 kann jede Mischung von IBM 3420-Laufwerken steuern.

6.15. Plattenstapel

In diesem Abschnitt wollen wir über die folgenden Typen von Plattenstapeln sprechen: IBM 1316, IBM 2316, IBM 3336 (Modell 1) und IBM 3336 (Modell 11). In der Abb. 6.38. ist ein Plattenstapel des Types IBM 3336 (Modell 1) dargestellt.

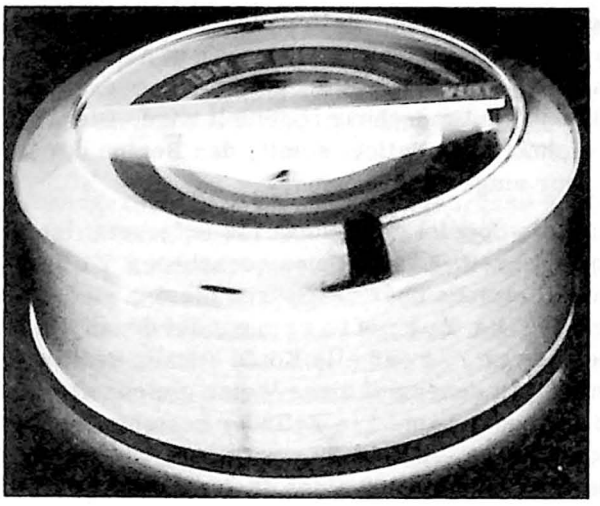


Abb. 6.38. Plattenstapel IBM 3336, Modell 1

Ein Plattenstapel setzt sich aus mehreren Aluminiumplatten zusammen, die alle starr mit einer zentralen Säule verbunden sind. Die Platten sind mit der gleichen Eisenoxydschicht überzogen, die bei Magnetbändern benutzt wird. Die Platten, über die wir diskutieren wollen, besitzen einen Durchmesser von 14 Zoll (35,56 cm) und sind im Abstand von etwa 0,4 Zoll (1,02 cm) voneinander montiert. Eine Abdeckhaube aus Plastik schützt die Platten, wenn der Stapel nicht in Gebrauch ist. Die Plattenstapel sind auswechselbar; sie entsprechen damit den Magnetbandrollen.

Die beiden Oberflächen jeder Platte werden als untere und obere bezeichnet (besser: Grundfläche und Deckfläche). Die Deckfläche der obersten und die Grundfläche der untersten Platte werden nicht benutzt, weil sie gegenüber Beschädigungen viel zu empfindlich sind. Der Plattenstapel IBM 1316 besteht aus sechs Platten; es sind also 10 Plattenflächen für die Aufzeichnung von Informationen verfügbar. Der Stapel IBM 2316 weist 11 Platten mit 20 nutzbaren Oberflächen auf. Der Plattenstapel IBM 3336 besteht ebenfalls aus 11 Platten, doch sind bei ihm nur 19 nutzbare Oberflächen verfügbar, weil eine Oberfläche für gewisse Synchronisierungsaufzeichnungen benötigt wird, die im Herstellungswerk aufgebracht und nie mehr geändert werden.

Die Oberflächen werden fortlaufend von 0 bis 9, 19 oder 18 numeriert. Die Grundfläche der obersten Platte trägt dabei die Nummer 0, die Deckfläche der nächsten Platte die Nummer 1, ihre Grundfläche die Nummer 2 usw.

Jede Plattenfläche enthält Spuren (konzentrische Kreise), auf die die Informationen aufgezeichnet werden können. Die Spuren werden von außen nach innen durchnumeriert, beginnend bei 0 und endend bei 202, 403 oder 807. Auf jeder Spur können, abhängig von den genannten Typen, maximal 3625, 7294 oder 13030 Bytes gespeichert werden.

Vom physikalischen Standpunkt aus gesehen, stellen die Spuren also Kreise auf den Plattenoberflächen dar. Um den für die Aufzeichnungen wichtigen Anfangspunkt einer Spur anzuzeigen, ist eine physikalische Markierung (Loch) auf der untersten Fläche des Plattenstapels eingraviert, die bei jeder Umdrehung abgefühlt wird. Dieser Anfangspunkt, auch als Indexpunkt bezeichnet, ermöglicht somit, den Beginn der Informationen festzustellen, die auf einer Spur aufgezeichnet sind.

Nachdem ein Plattenstapel auf die Spindel des Laufwerks aufgesetzt ist, kann die Schutzhülle entfernt werden. Ein mit Schreib/Lese-Köpfen versehener Zugriffsmechanismus bewegt sich während des Betriebs über die Spuren hinweg. Jeder Oberfläche ist ein Schreib/Lese-Kopf zugeteilt. Die Zugriffsarme , auf denen die Köpfe befestigt sind, werden stets zusammen bewegt, so daß alle Köpfe ständig senkrecht übereinander stehen. Die Menge aller Spuren, zu denen auf diese Weise gleichzeitig zugegriffen werden kann, wird als Zylinder bezeichnet. Ein Zylinder besteht also aus den gleichnumerierten Spuren aller Oberflächen. Abb. 6.39. zeigt uns, wie der Zugriffsmechanismus eines Laufwerks gerade den Zylinder 202 eines Plattenstapels IBM 1316 überstreicht. Die Köpfe selbst berühren die Oberflächen der Platten nicht; sie schweben vielmehr, von einem Luftpolster getragen, ein wenig über ihnen. Die Luftpolster entstehen durch die rasche Umdrehung der Platten.

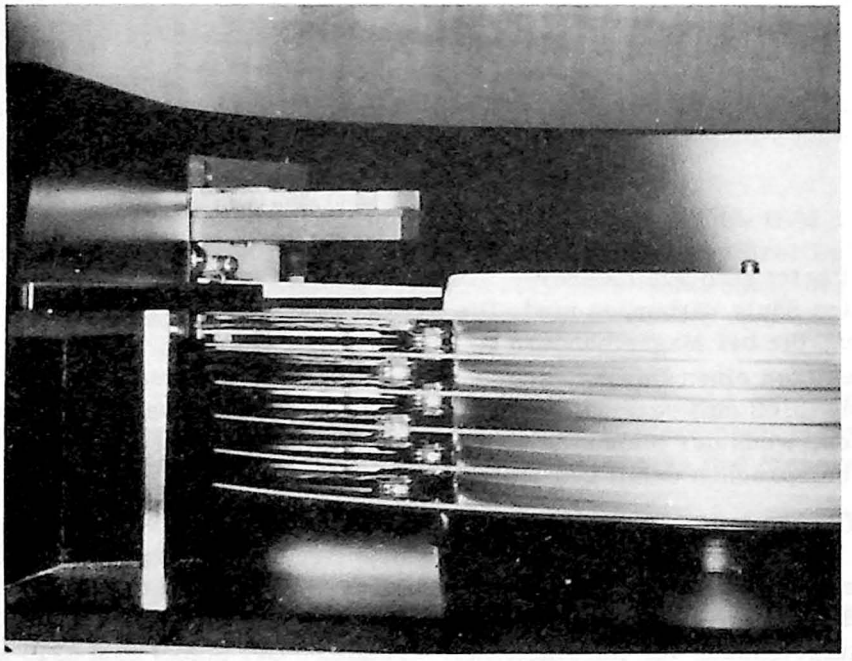


Abb. 6.39. Auf das Laufwerk eines Magnetplattenspeichergerätes aufgesetzter Plattenstapel IBM 1316

Wenn ein Stückchen eines Magnetbandes abgenutzt ist oder auf eine andere Art unbrauchbar geworden ist, so wird der Befehl ERG (Streichen Kluft) dazu benutzt, um über die schadhafte Bandstelle hinwegzuspringen. Bei einem Plattenspeicher scheidet ein solches Überspringen von vornherein aus. Deshalb schreibt man die Informationen, die an und für sich zu einer schadhaften Spur gehören, einfach auf eine intakte andere Spur. Die normalerweise verwendete Spur heißt Primärspur; die Spur, die verwendet wird, wenn die Primärspur fehlerhaft ist, wird Ersatzspur oder Ausweichspur genannt.

Gewöhnlich sind die Spuren der letzten drei Zylinder zum Gebrauch als Ersatzspuren reserviert. Selbst, wenn nur ein Teil einer Spur schadhaft ist, ist es gewöhnlich besser, die gesamte Spur unbenutzt zu lassen als zu versuchen, einen Teil derselben noch zu verwenden. Bei der Berechnung der Kapazität von Plattenstapeln werden die Ausweichspuren nicht mitgezählt.

Bei allen Plattenspeichergeräten sind die Bedienungsorgane ähnlich gestaltet. Der Schalter START-STOP sorgt für den Beginn der Rotation des Plattenstapels bzw. stoppt sie in der anderen Stellung ab. Es dauert fast eine Minute, bis der Plattenstapel die für seine Verwendungsfähigkeit erforderliche Endgeschwindigkeit erreicht hat. Neben der grünen Signallampe, die den betriebsbereiten Zustand des Gerätes anzeigt, ist außerdem eine rote Anzeigelampe vorhanden. Leuchtet diese auf, so liegt ein Hardwarefehler vor, der keine Plattenspeicheroperation mehr zuläßt. Einige Laufwerke haben zusätzlich einen Schalter, mit dem das Gerät gegenüber einer Benutzung gesperrt werden kann. Durch ihn wird die Meßeinrichtung ausgeschaltet; damit kann von der IBM /370 aus nicht mehr zugegriffen werden. Einige Geräte besitzen einen Schreib/Lese-Schalter. Wird dieser auf „Lesen" gestellt, so kann auf den Stapel nicht geschrieben werden. Die meisten Laufwerke besitzen am Gehäuse einen Stecker, mit dem die Adresse eingestellt werden kann. Der Stecker enthält nämlich die rechtsbündige Sedezimalziffer der Geräteadresse. Das Laufwerk geht erst dann in den betriebsbereiten Zustand, wenn ein Stecker eingesetzt ist. Ein Spezialstecker mit dem Buchstaben S wird vom Wartungspersonal benutzt.

Plattenstapel kosten in den USA zwischen 200 und 1000 Dollars, hoch im Vergleich zu den etwa 10 Dollars betragenden Kosten einer Magnetbandrolle. Plattenspeichergeräte haben den Vorteil, daß sie schneller als Magnetbandgeräte sind und, bei sehr kleinen Systemen, auch billiger. Wegen der Plattenstapelkosten werden jedoch große Datenmengen gewöhnlich auf Magnetbändern gespeichert.

6.16. Die Datenorganisation bei Speichereinheiten mit direkter Zugriffsmöglichkeit (SEDZ)

Wir können uns eine Platte als eine Ansammlung von Hunderten von Magnetbandstückchen vorstellen, jedes nur ein paar cm lang. Setzt man einen Plattenspeicher nur unter Realisierung dieses Gedankenguts ein, so nutzt man die wertvollen Eigenschaften eines Speichers mit direkter Zugriffsmöglichkeit nur recht ungenügend aus. Grundsätzlich sollten wir uns deshalb immer wieder die Tatsache vor Augen halten, daß wir zu jeder Spur unabhängig von jeder anderen Spur zugreifen können. Darüberhinaus ist es auch noch möglich, daß wir die Spuren selbst so einteilen können, daß der verfügbare Speicherplatz so vorteilhaft wie möglich ausgenutzt wird.

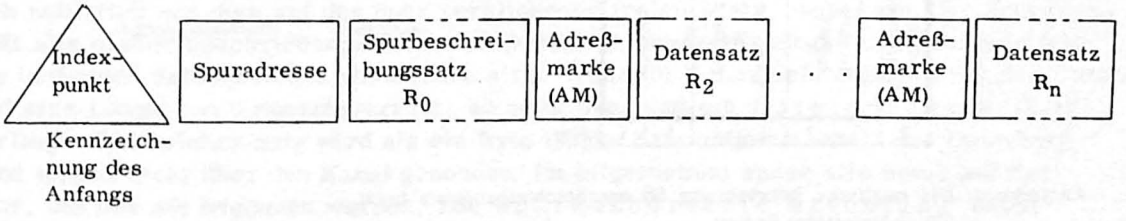
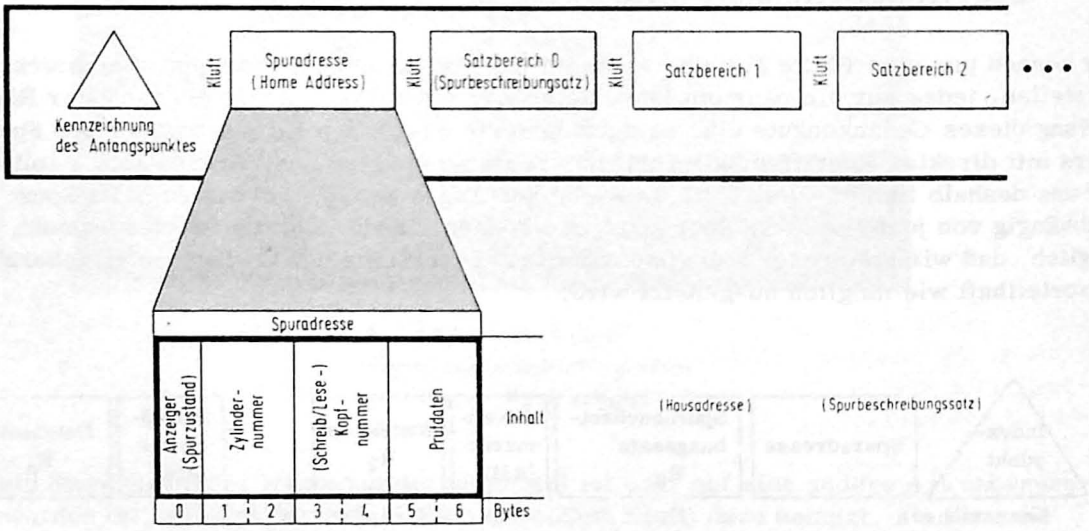


Abb. 6.40. Schematische Darstellung des Aufbaus von Spuren bei Speichereinheiten mit direkter Zugriffsmöglichkeit (SEDZ)

In der Abb. 6.40. ist der Aufbau von Spuren von Speichereinheiten mit direkter Zugriffsmöglichkeit (SEDZ) [1] schematisch dargestellt. Hinter dem Indexpunkt wird zunächst auf jede Spur eine Spuradresse , die sogenannte Hausadresse der Spur, aufgezeichnet. In dieser Adresse sind u.a. die Zylindernummer und die Nummer des Schreib/Lese-Kopfes enthalten; ferner stehen in ihr auch Informationen, die zu Prüfzwecken herangezogen werden. Hinter der Spuradresse folgt der sogenannte Spurbeschreibungssatz , den wir mit R_0 bezeichnen wollen. Alle Standardprogramme der IBM benutzen diesen Satz zur Speicherung von acht Datenbytes, die den Zustand der Spur, wie Belegung usw. festhalten. Danach kommt eine Adreßmarke, abgekürzt AM. Darunter versteht man ein spezielles Zeichen, das vor allen Satzbereichen (Regionen) mit Ausnahme des Satzbereiches von R_0 steht. An die erste Adreßmarke schließt sich eine Region an, die den ersten Datensatz, mit R_1 bezeichnet, aufnimmt. Daraufhin folgt wieder eine Adreßmarke und der nächste Datensatzbereich. Auf diese Weise ist eine Spur so oft gegliedert, bis der Indexpunkt eine weitere Wiederholung dieses Wechsels zwischen Adreßmarke und Satzbereich verhindert. Die Anzahl der Sätze, die auf eine Spur aufgezeichnet werden können, hängt allein von der Länge der Sätze ab. Jedem Satzbereich muß jedoch unbedingt eine Adreßmarke vorausgehen.

In den Spuren müssen die Bits eines Bytes hintereinander gespeichert werden. Infolgedessen würde die Speicherung des Paritätsbits, d.h. des neunten Bits eines Bytes, einen recht beachtlichen Platz beanspruchen. Die Steuereinheit befreit deshalb die ankommenden Bytes von den Paritätsbits und zeichnet nur die acht Bits, die zur Verschlüsselung der Zeichen dienen, auf. Parallel zum Entfernen der Paritätsbits werden jedoch von der Steuereinheit zwei Prüfbytes berechnet, von uns als zyklische Prüfdaten bezeichnet. Die Prüfdaten werden im Anschluß an die Informationsgruppen geschrieben, die auf eine Plattenspur aufgezeichnet werden. Wenn nun die Informationen wieder gelesen werden, so werden die Paritätsbits wieder regeneriert, die Prüfdaten neu errechnet und mit den aufgezeichneten Prüfdaten verglichen. Bei der IBM 3336 wird an Stelle der zyklischen Prüfdaten ein sogenannter Fehlerbeseitigungscode verwendet; seine Funktion ähnelt jedoch der der Prüfdaten.

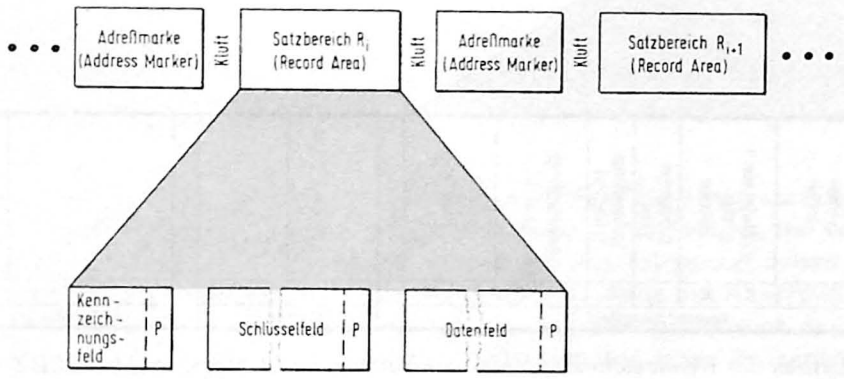


Anmerkung: Die englische Bezeichnung für Spurbeschreibungssatz heißt Track Descriptor Record

Abb. 6.41. Der Aufbau der Spuradresse (Hausadresse)

1) Im angloamerikanischen Sprachraum als „Direct Access Storage Devices", abgekürzt DASD, bezeichnet

Die Abb. 6.41. zeigt die Einzelheiten über den Aufbau der Spuradresse. Die Bits 0 bis 5 des Anzeigerbytes sind stets gleich 0, Bit 6 ist dann und nur dann gleich 1, wenn die Spur schadhaft ist. Die Bitstelle 7 dagegen enthält dann und nur dann eine 1, wenn die betreffende Spur als Ersatzspur benutzt wird. Zum Schreiben der Spuradressen auf einen neuen Plattenstapel benutzt man im allgemeinen ein IBM-Dienstprogramm. Hiervon ausgenommen sind die IBM 3336. Bei ihnen sind die Hausadressen bereits werksseitig aufgezeichnet. Man beachte, daß für die duale Zylindernummer zwei Bytes bereitgestellt werden. Das gleiche gilt für die Schreib/Lese-Kopf-Nummern. Jede Spuradresse umfaßt somit sieben Bytes, wovon zwei für die Prüfdaten bestimmt sind.



Anmerkung: P bedeutet hier Prüfdaten

Abb. 6.42. Der Aufbau der Satzbereiche (Regionen) bei Speichereinheiten mit direkter Zugriffsmöglichkeit (SEDZ)

Die Abb. 6.42. zeigt schematisch Einzelheiten über den Aufbau eines Satzbereiches (einer Region) bzw. über die Zusammensetzung der in ihm gespeicherten Sätze. Jeder Satz kann tatsächlich aus drei getrennten Teilen, die Felder genannt werden, zusammengestellt sein. Zunächst kommt das sogenannte Kennzeichnungsfeld, danach das Schlüsselfeld und schließlich das Datenfeld. Das Kennzeichnungsfeld enthält neben Informationen, die zur Identifizierung des betreffenden Satzes herangezogen werden, auch Angaben über die Länge der beiden anderen Felder. Das Schlüsselfeld kann eine Länge zwischen 00 und FF (sedezimal) Bytes besitzen, zwischen 0 und 255 Bytes also. Eine Länge von 0 bedeutet selbstverständlich, daß kein Schlüsselfeld vorhanden ist. Daraus erkennt man deutlich, daß das Schlüsselfeld nicht notwendig bei einem Satz existieren muß. Man sagt deshalb auch, daß ein Satz mit Schlüssel bzw. ohne Schlüssel aufgezeichnet ist, je nachdem, ob das Schlüsselfeld vorhanden oder nicht vorhanden ist. Das Datenfeld kann eine Länge zwischen 0000 und FFFF (sedezimal) Bytes, d.h. zwischen 0 und 65535 Bytes, aufweisen. Die maximal mögliche Datenfeldlänge bestimmt sich natürlich aus dem auf der Spur verbliebenen freien Platz, wobei von der Spurkapazität alle bisher beschriebenen Satzbereiche sowie Kennzeichnungs- und Schlüsselfeld des laufenden Satzbereiches abzuziehen sind. Wenn im Kennzeichnungsfeld für das Datenfeld eine Länge von 0 spezifiziert ist, so heißt das, daß ein Dateiendesatz (EOF) vorliegt. Ein solcher Satz wird als ein Byte langer Satz aufgezeichnet; das Datenbyte wird jedoch nicht über den Kanal gesendet. Im allgemeinen enden alle Sätze auf der Spur, auf der sie begonnen wurden. Die Spurenüberlaufeinrichtung (auch Spurenverbindung genannt) ermöglicht jedoch die Fortsetzung der Aufzeichnung eines Satzes auf der nächstfolgenden Spur; eine Zylindergrenze darf dabei jedoch nicht überschritten werden. Abgesehen von den beiden Bytes, die die Prüfdaten aufnehmen,

und den EOF-Daten, enthalten sowohl das Schlüsselfeld als auch das Datenfeld nur die Daten, die von Programmen in sie hineingestellt wurden.

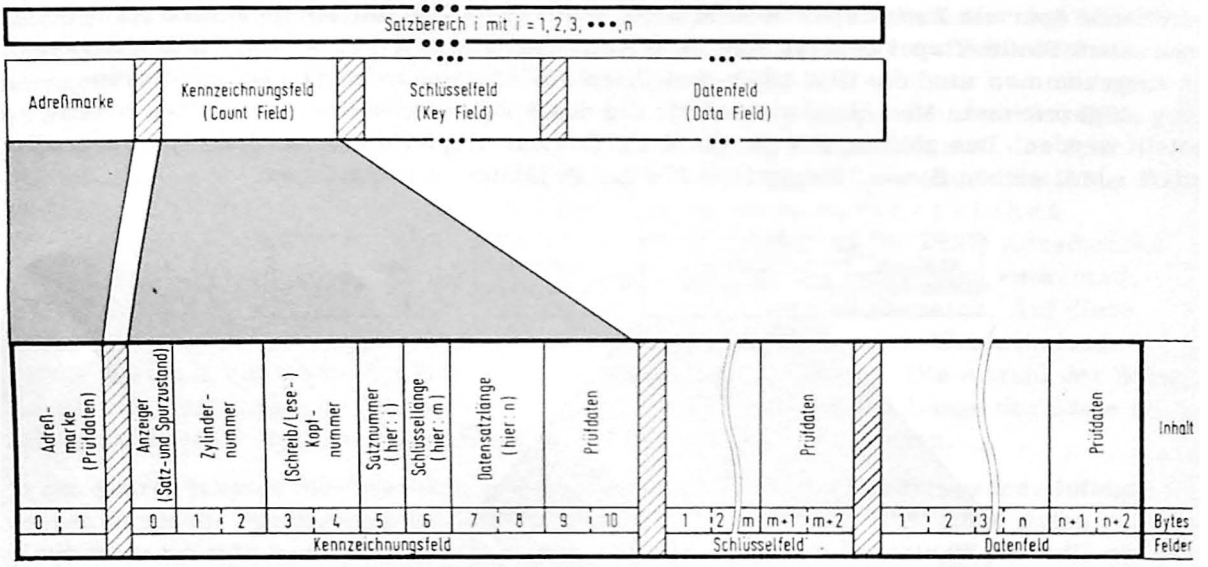


Abb. 6.43. Der Aufbau des Kennzeichnungsfeldes von Satzbereichen (Regionen) bei SEDZ

Die Abb. 6.43. zeigt die Einzelheiten über den Aufbau des Kennzeichnungsfeldes. Im Anzeigerbyte enthält die Bitstelle 0 dann und nur dann eine 1, wenn es sich um die Satzbereiche mit gerader Ordnungsnummer handelt, also um R_0, R_2 usw. Auf der Bitstelle 1 ist dann und nur dann eine 1 enthalten, wenn es sich um Spurenüberlaufsätze handelt, die auf der nachfolgenden Spur fortgesetzt werden. Den Bitstellen 2 bis 7 sind die gleichen Dualziffern zugewiesen, die im Anzeigerbyte der Spuradresse auf den gleichen Stellen stehen. Das Anzeigerbyte des Kennzeichnungsfeldes kann niemals durch ein Programm geschrieben oder gelesen werden; es wird vielmehr während des Schreibens durch die Steuereinheit erzeugt, bei einer Leseoperation benutzt es die Steuereinheit für ihre eigenen Zwecke. Genau wie die Prüfdaten wird das Anzeigerbyte niemals zum Hauptspeicher übertragen.

Wenn zu einem Satz in einer intakten Primärspur zugegriffen wird, so müssen die Zylinder- und die Kopfnummern in den Sätzen mit der Zylinder- und der Kopfnummer in der Spuradresse übereinstimmen. Bei in Benutzung befindlichen Ersatzspuren stehen in der Spuradresse die zur Spur gehörigen Zylinder- und Kopfnummern; in den Kennzeichnungsbereichen der Sätze stehen jedoch die Zylinder- und Kopfnummern, die zu der Primärspur gehören, von der die betreffenden Sätze ausgelagert wurden. Auf den schadhaften Primärspuren enthält die Spuradresse ebenfalls die eigene Zylinder- und Kopfnummer; in den Kennzeichnungsfeldern der Sätze sind jedoch die Zylinder- und die Kopfnummer der Ersatzspur aufgezeichnet, in die die betreffenden Sätze aufgelagert wurden. Wegen dieser Zusammenhänge bezeichnet man oft die Spuradresse auch als Hausadresse.

Der Satzbereich R_0, der bekanntlich den sogenannten Spurbeschreibungssatz enthält, unterscheidet sich physikalisch gesehen nur dadurch von den übrigen Satzbereichen, daß ihm keine Adreßmarke vorangeht. Dieser Satzbereich ist immer für die Speicherung eines aus acht Bytes bestehenden Datenfeldes reserviert; ein Schlüsselfeld ist niemals für diesen Satzbereich vorgesehen. Im Spurbeschreibungssatz stehen Informationen über die Belegung der betreffenden Spur. In die in Abschnitt 6.15. bereits erwähnten Spurenkapazitäten ist der Spurbeschreibungssatz nicht mit eingeschlossen.

Zwischen den einzelnen aufgezeichneten Informationsgruppen befindet sich jeweils eine Kluft, d.h. Stellen, die frei von Informationen jeglicher Art sind. Die Länge der Kluft, die das Schlüsselfeld vom nachfolgenden Datenfeld des gleichen Satzes trennt, beträgt immer 18 Bytes. Die anderen Klüfte variieren in ihrer Länge, wobei diese von der Länge der zuvor aufgezeichneten Informationsgruppe abhängt. Die Klüfte sind wegen der mechanischen Schwierigkeiten, die Plattenumdrehungen exakt konstant zu halten, notwendig. Anhang H enthält Formeln zur Berechnung des für einen Satz erforderlichen Platzes für die am häufigsten eingesetzten Plattenspeichergeräte.

6.17. Die Befehlsschlüssel bei den Speichereinheiten mit direkter Zugriffsmöglichkeit (SEDZ)

Im Gegensatz zu den meisten anderen EA-Geräten erfordert die Programmierung bei den SEDZ [1] die Befehlskettung; außerdem ist der Befehl „Verzweigen im Kanal (Transfer in Channel)" notwendig. Wie wir bereits im vorigen Kapitel gesagt haben, wird nämlich das nächste Kanalbefehlswort (CCW) übergangen, wenn bei Abschluß einer Operation das Gerät sowohl der Bitstelle für die Statusänderung als auch der Bitstelle für Geräteende eine 1 zugewiesen hat. Das geschieht aber bei einer Reihe von Befehlen.

Wir können die Befehle in folgende sechs Gruppen einteilen:

1. *Suchen* oder *Einstellen (Seek)*
2. *Aufsuchen* oder *Durchsuchen (Search)*
3. *Lesen (Read)*
4. *Schreiben* oder *Aufzeichnen (Write)*
5. *Steuerung (Control)*
6. *Verschiedenes (Miscellaneous)*

Es gibt die folgenden drei Suchbefehle:

a) 07 —— Suchen (Seek)
b) 0B —— Suchen Zylinder (Seek Cylinder)
c) 1B —— Suchen Kopf (Seek Head)

Bei Suchbefehlen wird ein sechs Bytes langes Feld aus dem Hauptspeicher übertragen (Abb. 6.44.).

Bytes	0	1	2	3	4	5
Inhalt der Bytes	Zelle		Zylinder		Schreib- und Lesekopf	

Abb. 6.44. Übertragene Bytes bei den Suchbefehlen

Jedes aus zwei Bytes bestehende Unterfeld nimmt einen Dualwert auf. Bei den bisher besprochenen Plattenspeichern müssen die Bytes 0 und 1 Nullen enthalten. Die Bytes 2 und 3 enthalten die Zylindernummer; Byte 4 enthält wiederum Null und Byte 5 die Kopfnummer.

1) Im angloamerikanischen Sprachraum als „Direct Access Stoage Devices, abgekürzt DASD, bezeichnet.

Die Suchbefehle bewirken, daß der Zugriffsmechanismus auf den angegebenen Zylinder positioniert und der angegebene Kopf ausgewählt wird.

Bei Befehl „Suchen Zylinder " werden die beiden Bytes 0 und 1 vernachlässigt. Bei den bisher erwähnten Speichern ist dieser Befehl identisch dem Befehl „Suchen" mit dem Befehlsschlüssel 07.

Der Befehl „Suchen Kopf" ignoriert die Bytes 0 bis 3. Er ruft keine Bewegung der Arme hervor, sondern wählt vielmehr nur einen anderen Kopf und damit auch eine andere Spur aus. Obwohl die Bytes 0 bis 3 nicht benutzt werden, müssen sie eine gültige Adresse aufweisen.

Wenn der Kanal mehr als sechs Bytes zur Steuereinheit sendet, so wird eine falsche Satzlänge (Wrong Length Record, abgekürzt WRL) angezeigt und nur die ersten sechs Bytes werden verwendet. Wenn weniger als sechs Bytes übergeben werden, so werden diejenigen Bitstellen auf 1 gesetzt, die Befehlszurückweisung und Suchfehler anzeigen; die Operation wird also nicht ausgeführt. Die gesendeten sechs Bytes müssen eine gültige Adresse bilden, anderenfalls kommt es zur gleichen, eben geschilderten Fehlersituation.

Die Befehle beim Durchsuchen vergleichen im Hauptspeicher stehende Bytes mit Bytes auf der SEDZ. Es gibt folgende Durchsuchbefehle:

a) 39 —— Durchsuchen auf gleiche Spuradresse (Search Home Address Equal)
Die Bytes 1 bis 4 der Spuradresse werden mit den Daten aus dem Hauptspeicher verglichen. Die Anzeiger und die Prüfdaten werden hiervon nicht berührt. Wenn mehr als vier Bytes vom Hauptspeicher gesendet werden, so werden nur die ersten vier Bytes verwendet. Wenn weniger als vier Bytes vom Hauptspeicher ankommen, so erstreckt sich der Vergleich nur über die entsprechende Anzahl von Bytes auf dem Plattenspeicher. Tritt die Geräteende-Bedingung auf, so wird auch die Bitstelle für die Statusänderung auf 1 gesetzt, wenn der Vergleich erfolgreich war. Die Ausführung dieses Befehls beginnt, sobald der Indexpunkt festgestellt ist; die Ausführung wird am Ende der Spuradresse abgeschlossen.

b) 31 —— Durchsuchen auf gleiche Kennzeichnung (Search Identification Equal)

c) 51 —— Durchsuchen auf Kennzeichnung hoch (Search Identification High)

d) 71 —— Durchsuchen auf Kennzeichnung gleich oder hoch (Search Identification Equal or High)
Die Bytes 1 bis 5 des Kennzeichnungsfeldes werden mit den Daten verglichen, die vom Hauptspeicher übertragen wurden. Der Rest des Kennzeichnungsfeldes, nämlich das Byte 0 und die Bytes 6 bis 10, sind in diese Operation nicht verwickelt. Wenn mehr als 5 Bytes gesendet werden, so werden nur die ersten fünf Bytes zum Vergleich herangezogen; werden weniger als fünf Bytes gesendet, so wird der Vergleich beendet, wenn die gleiche Anzahl von Bytes vom Plattenspeicher in den Vergleich eingegangen ist. Beim Auftreten der Geräteende-Bedingung wird auch gleichzeitig die Bitstelle für Statusänderung auf 1 gesetzt, wenn der Vergleich erfolgreich war, d.h. die Hauptspeicherdaten sind gleich den entsprechenden Daten aus dem Kennzeichnungsfeld (Befehlsschlüssel 31), größer als die Daten aus dem Kennzeichnungsfeld (Schlüssel 51) bzw. größer oder gleich den Daten aus dem Kennzeichnungsfeld (71). Die Ausführung dieses Befehls beginnt beim ersten Kennzeichnungsfeld, das nach dem Indexpunkt oder nach einer Adreßmarke festgestellt wurde. Wird dabei auf den Indexpunkt gestoßen, so wird nach R_0 gesucht. Die Operation wird am Ende des Kennzeichnungsfeldes abgeschlossen.

e) 29 —— Durchsuchen auf gleichen Schlüssel (Search Key Equal)

f) 49 —— Durchsuchen auf Schlüssel hoch (Search Key High)

g) 69 —— Durchsuchen auf Schlüssel gleich oder hoch (Search Key Equal or High)

Die vom Hauptspeicher empfangenen Daten werden mit dem Schlüsselfeld des Plattenspeichers verglichen. Die Operation wird dann beendet, wenn entweder der Kanal das Übertragen der Daten aus dem Hauptspeicher abbricht oder das Schlüsselfeld ausgeschöpft ist. Außer der Bitstelle für Geräteende wird auch der Bitstelle für die Statusänderung eine 1 zugewiesen, wenn der Vergleich zum Ergebnis „gleich" (Befehlsschlüssel 29), „hoch" (Befehlsschlüssel 49), d.h. die Hauptspeicherdaten sind größer als die Daten aus dem Schlüsselfeld, bzw. „gleich oder hoch" (69), führte. Die Bitstelle für die Statusänderung wird jedoch nicht auf 1 gesetzt, wenn das Schlüsselfeld die Länge 0 besitzt. Die Ausführung dieses Befehls beginnt mit dem ersten Schlüsselfeld, auf das infolge der Plattenumdrehung gestoßen wird. R_0 bildet hierbei eine Ausnahme: Dieser Satz wird nicht durchsucht, außer wenn der Befehl „Durchsuchen auf Schlüssel" an einen Befehl „Durchsuchen auf Kennzeichnung" gekettet ist, der eine Durchsuchung von R_0 zur Folge hatte. Die Ausführung der Befehle „Durchsuchen auf Schlüssel" wird am Ende des Schlüsselfeldes abgeschlossen.

Die Bitstelle „Kein Satz gefunden" des Prüfbytes 1 wird auf 1 gesetzt und verursacht dadurch einen „Gerätefehler", wenn während einer Folge von geketteten Befehlen ohne dazwischen liegende Lese- oder Schreibbefehle der Indexpunkt zweimal den Zugriffsmechanismus passiert. Die Feststellung eines Gerätefehlers bewirkt wiederum, daß die Befehlskettung beendet wird. Diese Eigenschaft, zusammen mit einem Befehl „Verzweigen im Kanal" kann dazu verwendet werden, um eine ganze Spur nach einem Satz mit einer gegebenen Kennzeichnung, einem gegebenen Schlüssel oder gegebenem Schlüssel und gegebenen Daten zu durchsuchen. Um z.B. den Satz R_3 auf der Spur 4 des Zylinders 18 zu lesen, können wir die Befehlsfolge aufsetzen, die in Abb. 6.45. dargestellt ist.

```
002 000     07 003 000 40 000 006   SUCHEN (EINSTELLEN)
002 008     31 003 002 40 000 005   DURCHSUCHEN AUF GLEICHE KENNZEICHNUNG
002 010     08 002 008 00 000 000   VERZWEIGEN IM KANAL (TRANSFER IN CHANNEL)
002 018     -- (LESEBEFEHL) --

003 000  BIS  003 006               0000 0012 0004 03   ZYLINDER/KOPF/SATZ
```

Abb. 6.45. Befehlsfolge zum Aufsuchen eines Satzes auf einer Spur

Der Suchbefehl führt uns zunächst auf die passende Spur. Der erste Durchsuchbefehl gilt für den ersten Satz, der dem Indexpunkt oder der ersten Adreßmarke folgt, auf die der Kopf infolge der Plattenumdrehung stößt. Ist der erste Durchsuchbefehl nicht erfolgreich, so führt der Befehl „Verzweigen im Kanal" zurück zum Durchsuchbefehl, der somit wiederholt wird. Wenn gegebenenfalls der Durchsuchbefehl erfolgreich abgeschlossen ist, wird der Verzweigungsbefehl übergangen, weil die Bitstelle für die Statusänderung im Gerätestatusbyte auf 1 gesetzt ist. Gibt es auf der ausgewählten Plattenspur jedoch nicht den spezifizierten Satz, so wird nach zwei Plattenumdrehungen ein Gerätefehler („kein Satz gefunden") angezeigt und damit die Befehlskettung beendet.

Befehle für das Durchsuchen und das Lesen können so aufgesetzt werden, daß sie bei der angegebenen Spur beginnen und mit dem Durchsuchen auf anschließenden Spuren des gleichen Zylinders fortfahren, bis das Ende des Zylinders erreicht ist. Für diese Mehrspuroperationen müssen die Durchsuchbefehle immer so gegeben werden, daß die Bitstelle, die die Mehrspuroperation anzeigt, auch wirklich eine 1 enthält. Immer, wenn beim Durchsuchen dem Indexpunkt begegnet wird, geht das Durchsuchen mit der nächsten Spur weiter. Die Bedingung „kein Satz gefunden" wird niemals gesetzt; der Bitstelle „Zylinderende" wird jedoch eine 1 zugewiesen, wenn der Indexpunkt der letzten Spur des Zylinders passiert ist. Dadurch wird „Geräteende" signalisiert und das Durchsuchen ohne Erfolg beendet. Die Mehrspuroperationen besitzen Befehlsschlüssel, die den Befehlsschlüsseln der anderen Lese- und Durchsuch-

befehle bis auf ein Bit gleichen: Die am weitesten links liegende Bitstelle des Befehlsschlüssels enthält bei allen Mehrspuroperationen eine 1. Damit ergeben sich die folgenden Befehlsschlüssel:

a) B9 —— Mehrspuriges Durchsuchen auf gleiche Spuradresse (Multiple Track Search Home Adress Equal)

b) B1 —— Mehrspuriges Durchsuchen auf gleiche Kennzeichnung (MT Search Identification Equal)

c) D1 —— Mehrspuriges Durchsuchen auf Kennzeichnung hoch (MT Search Id High)

d) F1 —— Mehrspuriges Durchsuchen auf Kennzeichnung gleich oder hoch (MT Search Id Equal or High)

e) A9 —— Mehrspuriges Durchsuchen auf gleichen Schlüssel (MT Search Key Equal)

f) C9 —— Mehrspuriges Durchsuchen auf Schlüssel hoch (MT Search Key High)

Zum Aufsuchen eines Satzes mit dem (sedezimalen) Schlüssel F0F3F4F6 im Zylinder 18 können wir die in Abb. 6.46. dargestellte Befehlsfolge einsetzen.

```
002 000     07 003 000 40 000 006    SUCHEN BZW. EINSTELLEN
002 008     39 003 002 40 000 004    DURCHSUCHEN AUF SPURADRESSE
002 010     A9 003 006 40 000 004    DURCHSUCHEN AUF GLEICHEN SCHLUESSEL (MS)
002 018     08 002 010 00 000 000    VERZWEIGEN IM KANAL
002 020     -- (LESEBEFEHL) --

003 000  BIS  003 005                0000 0012 0000
003 006  BIS  003 009                F0F3 F4F6   (SCHLUESSEL)
```

Abb. 6.46. Beispiel für eine Befehlsfolge zum Aufsuchen eines Satzes über den Schlüssel

In der Abb. 6.46. haben wir durch MS eine Mehrspuroperation angezeigt.

Wenn die Sätze mit Schlüsseln gespeichert werden, so werden sie oft so angeordnet, daß die Schlüssel in aufsteigender Reihenfolge vorkommen. Damit ist es aber sinnvoll, den Befehl „Durchsuchen auf Schlüssel gleich oder hoch" einzusetzen. Ergibt sich nämlich während der Ausführung der Operationen, daß das Schlüsselfeld auf dem Plattenspeicher einen größeren Wert enthält, so kann auf dem Plattenspeicher kein Satz existieren, der den gewünschten Schlüssel besitzt. Das Suchen kann somit abgestoppt werden, ohne daß es notwendig ist, den Rest des Zylinders auch noch zu durchsuchen.

Die Lesebefehle übertragen Daten vom Plattenspeicher zum Hauptspeicher. Da es nicht erforderlich ist, sie selbst zu ketten, werden sie gewöhnlich an einen Durchsuchbefehl gekettet. Es gibt die folgenden Lesebefehle:

a) 12 —— Lesen der Kennzeichnung (Read Count)

b) 92 —— Mehrspuriges Lesen der Kennzeichnung (MT Read Count)
Die Bytes 1 bis 8 des Kennzeichnungsfeldes werden durch dieses Befehle zum Kanal übertragen, das Anzeigerbyte und die Prüfdaten jedoch nicht. Es erfolgt das Lesen der Kennzeichnung desjenigen Satzes, der hinter der nächsten Adreßmarke aufgezeichnet ist. Der Satz R_0 kann durch diese Befehle nicht gelesen werden.

c) 1E —— Lesen der Kennzeichnung, des Schlüssels und der Daten (Read Count, Key and Data)

d) 9E —— Mehrspuriges Lesen der Kennzeichnung, des Schlüssels und der Daten (MT Read Count, Key and Data)

Die Bytes 1 bis 8 des Kennzeichnungsfeldes, der Inhalt des gesamten Schlüsselfeldes und der Inhalt des gesamten Datenfeldes werden zum Kanal übertragen. Es wird der erste Satz nach der ersten begegneten Adreßmarke gelesen. Der Spurbeschreibungssatz R_0 kann durch diesen Befehl nicht gelesen werden. Beim MS-Befehl wird der Satz R_1 der nächsten Spur gelesen, falls der Indexpunkt überschritten wird.

e) 0E —— Lesen des Schlüssels und der Daten (Read Key and Data)

f) 8E —— Mehrspuriges Lesen des Schlüssels und der Daten (MT Read Key, and Data)
Schlüsselfeld- und Datenfeldinhalt werden zum Kanal übertragen. Wenn gerade ein Kennzeichnungsfeld gelesen wurde, so erstreckt sich das Lesen von Schlüsselfeld und Datenfeld auf den gleichen Satz, von dem zuvor das Kennzeichnungsfeld gelesen wurde. Im anderen Fall erfolgt das Lesen der spezifizierten Teile desjenigen Satzes, der auf das nächste Kennzeichnungsfeld folgt.

g) 06 —— Lesen der Daten (Read Data)

h) 86 —— Mehrspuriges Lesen der Daten (MT Read Data)
Der Inhalt eines Datenfeldes wird zum Kanal übertragen. Es wird das Datenfeld des gleichen Satzes gelesen, wenn der Befehl in dem Augenblick gegeben wird, in dem sich der Lese/Schreibkopf gerade in der Kluft nach dem Schlüsselfeld oder nach dem Kennzeichnungsfeld eines Satzes befindet. In allen anderen Fällen wird jedoch das Datenfeld gelesen, das sich an das erste festgestellte Schlüsselfeld oder Kennzeichnungsfeld anschließt. Wenn die Schlüsselfeldlänge gleich Null ist, so sind die Befehle mit den Befehlsschlüsseln 0E und 06 bzw. 8E und 86 einander äquivalent.

i) 16 —— Lesen des Spurbeschreibungssatzes (Read R_0)

j) 96 —— Mehrspuriges Lesen des Spurbeschreibungssatzes (MT Read R_0)
Der gesamte Spurbeschreibungssatz wird zum Kanal übertragen. Diese Befehle sind also den Befehlen mit den Befehlsschlüsseln 1E und 9E identisch mit Ausnahme der Tatsache, daß eben der Spurbeschreibungssatz R_0 gelesen wird. Wenn diese Befehle an einen Durchsuch- oder Lesebefehl für die Spuradresse gekettet sind, so wird R_0 sofort gelesen; anderenfalls wird R_0 erst dann gelesen, wenn der Indexpunkt entdeckt wurde.

k) 1A —— Lesen der Spuradresse (Read Home Address)

l) 9A —— Mehrspuriges Lesen der Spuradresse (MT Read Home Address)
Die ersten fünf Bytes der Spuradresse, also einschließlich des Bytes, das die Anzeiger enthält, werden zum Kanal übertragen. Die Operation beginnt in dem Augenblick, in dem der Indexpunkt erreicht ist; sie wird beendet, wenn das Ende der Spuradresse festgestellt wird.

m) 02 —— Lesen der Anfangsspur (Read IPL)
Durch diesen Befehl wird zunächst der Zugriffsmechanismus auf die Spur 0 des Zylinders 0 eingestellt, danach wird der Satz R_1 zum Kanal übertragen. Im allgemeinen besteht R_1 aus 24 Bytes, die zum einleitenden Programmladen (IPL) herangezogen werden.

Die Schreibbefehle empfangen die Daten vom Kanal und schreiben die entsprechenden Sätze auf den Plattenspeicher. Im allgemeinen müssen sie an spezielle Befehle gekettet werden, damit sie richtig ausgeführt werden können. Mehrspuroperationen sind bei den Schreibbefehlen nicht erlaubt. Es gibt die folgenden Schreibbefehle:

a) 19 —— Schreiben der Spuradresse (Write Home Address)
Es werden die ersten fünf Bytes in die Spuradresse geschrieben. Wenn weniger als

fünf Bytes vom Kanal empfangen werden, so werden Nullen ergänzt. Werden dagegen mehr als fünf Bytes empfangen, so bleiben die überzähligen unbeachtet. Die Ausführung beginnt, sobald der Indexpunkt festgestellt wird. Der Befehl ist an den Befehl „Setzen Dateimaske" zu ketten. Er sollte für die Kennzeichnung defekter Spuren reserviert bleiben.

b) 15 —— Schreiben des Spurbeschreibungsssatzes (Write R_0)
Es wird der gesamte Spurbeschreibungssatz geschrieben. Acht Bytes werden ins Kennzeichnungsfeld geschrieben, die Länge des Schlüsselfeldes bzw. des Datenfeldes wird aus den entsprechenden Bytes des Kennzeichnungsfeldes entnommen. Wenn nicht genügend Daten vom Hauptspeicher gesendet werden, so ergänzt die Steuerung Nullen; werden zu viele Daten übertragen, so bleiben die überzähligen unbeachtet. Die Operation kann dann beginnen, wenn der Schreib/Lese-Kopf die Spuradresse passiert hat. Sie endet, wenn der Spurbeschreibungssatz geschrieben ist. Dieser Befehl muß entweder an den Befehl „Schreiben Spuradresse" oder an den Befehl „Durchsuchen auf Spuradresse gleich" gekettet werden. Wird diese Vorschrift nicht eingehalten, so wird der Bitstelle für ungültige Folge im Prüfbyte 1 eine 1 zugewiesen.

c) 1D —— Schreiben der Kennzeichnung des Schlüssels und der Daten (Write Count, Key and Data)
Dieser Befehl ist identisch dem Befehl mit dem Befehlsschlüssel 15 (siehe unter b); zusätzlich wird jedoch noch eine Adreßmarke vor dem eigentlichen Satz geschrieben. Dieser Befehl muß an einen Befehl „Schreiben des Spurbeschreibungssatzes" oder an einen gleichen Befehl gekettet werden. Er kann aber auch an einen erfolgreichen Befehl „Durchsuchen auf gleiche Kennzeichnung" oder „Durchsuchen auf gleichen Schlüssel" gekettet werden; der Vergleich muß sich hierbei jedoch über das gesamte Feld erstreckt haben. Zwischen dem Durchsuchen und dem Schreiben kann außerdem einer der folgenden Befehle liegen: Lesen des Schlüssels und der Daten (0E), Lesen der Daten (06).

d) 01 —— Spezielles Schreiben der Kennzeichnung, des Schlüssels und der Daten (Write Special Count, Key, and Data)
Dieser Befehl unterscheidet sich nur dadurch von dem unter c) beschriebenen Befehl, daß er außerdem eine 1 in die Bitstelle 1 des Anzeigerbytes stellt. Dieser Befehl kann daher nur dann verwendet werden, wenn die Spurenüberlaufeinrichtung installiert ist. Bei einigen Plattenspeichergeräten kann dieser Befehl nicht benutzt werden.

e) 05 —— Schreiben der Daten (Write Data)
Es wird nur das Datenfeld neu geschrieben. Kennzeichnungsfeld und Schlüsselfeld bleiben unzerstört erhalten. Die Anzahl der zu schreibenden Bytes ist durch die Datenlänge festgelegt, die im Kennzeichnungsfeld angegeben ist. Falls es nötig ist, werden zur Auffüllung des Feldes sedezimale Nullen ergänzt. Dieser Befehl muß durch Kettung von einem vorangehenden erfolgreich ausgeführten Befehl „Durchsuchen auf gleiche Kennzeichnung" oder „Durchsuchen auf gleichen Schlüssel" erreicht werden. Der Vergleich muß sich dabei über das volle Feld erstreckt haben. Im anderen Fall wird nämlich der Bitstelle 3 des Prüfbytes 1 (ungültige Folge) eine 1 zugewiesen.

f) 0D —— Schreiben des Schlüssels und der Daten (Write Key, and Data)
Dieser Befehl bewirkt das gleiche wie der unter e) beschriebene Befehl; zusätzlich wird außerdem das Schlüsselfeld beschrieben. Dieser Befehl muß durch Kettung über einen Befehl „Durchsuchen auf gleiche Kennzeichnung" erreicht werden.

Es wäre noch zu erwähnen, daß die Kennzeichnung,der Schlüssel und die Daten, die zu einem kompletten Satz gehören, aneinanderstoßende Hauptspeicherplätze belegen müssen, wenn sie in eine Schreiboperation verwickelt sind, die alle drei Felder anspricht. Dasgleiche gilt für Schlüsselfeld und Datenfeld, sofern sie in eine Schreiboperation ein-

gehen, in die beide Felder verwickelt sind. Bei den Leseoperationen werden die Inhalte der Plattenspeicherfelder ebenfalls in aneinandergrenzende Hauptspeicherplätze übertragen.

Die Spurenüberlaufeinrichtung ermöglicht es, daß die Sätze von einer Spur in die nächste übergehen, allerdings nur innerhalb eines Zylinders. Wenn bei Lesebefehlen die Bitstelle 1 des Anzeigerbytes im Kennzeichnungsfeld eine 1 enthält, so wird am Ende des Datenfeldes das Lesen automatisch auf den Satzbereich R_1 der nächsten Spur ausgedehnt. Der unter d) beschriebene Befehl für das spezielle Schreiben kann nur auf den letzten Satz angewendet werden, der in eine Spur geschrieben werden soll. Dieser Befehl wird für jeden Teil eines Überlaufsatzes mit Ausnahme des letzten Teiles benötigt. Für den letzten Teil ist ein normaler Schreibbefehl (Schreiben der Kennzeichnung, des Schlüssels und der Daten) zu geben. Das Fehlen einer 1 auf der Bitstelle 1 der Anzeigerbytes signalisiert umgekehrt beim Lesen, daß das letzte Satzsegment übertragen wurde.

Es gibt die folgenden Steuerungsbefehle:

a) 11 —— Löschen (Erase)
Alles, was auf dem Rest einer Spur bis zum Indexpunkt aufgezeichnet ist, wird gelöscht. Dieser Befehl zerstört die Adreßmarken, die sonst auf den Spuren hinter Sätzen stehen bleiben würden, von denen erwartet wird, daß sie vom Plattenspeicher entfernt sind. Gewöhnlich gibt man diesen Befehl nach einem Spurenüberlauf, d.h., wenn man versucht hat, einen Satz zu schreiben, der in den noch verfügbaren Speicherplatz auf der Spur nicht mehr hineinpaßt. Für diesen Befehl sind die gleichen Kettungsvorschriften zu beachten wie beim Befehl „Schreiben der Kennzeichnung , des Schlüssels und der Daten".

b) 03 —— Keine Operation (Nop)
Die Bitstellen für Kanalende und Einheitenende werden nach Interpretation dieses Befehls auf 1 gesetzt.

c) 1F —— Setzen der Dateimaske (Set File Mask)
Es wird ein Byte zur Steuereinheit gesendet, das jedoch nicht weiter zur SEDZ übertragen wird. Die in diesem Byte stehenden Bits erlauben oder verhindern gewisse Befehlstypen. Die Bitstellen 2, 5, 6 und 7 haben dabei keine Bedeutung. Die Bits auf den Bitstellen 0 und 1 erlauben bzw. verhindern die Schreibbefehle wie folgt:

00	Es sind alle Schreibbefehle erlaubt mit Ausnahme von „Schreiben der Spuradresse" und „Schreiben des Spurbeschreibungssatzes".
01	Es sind überhaupt keine Schreibbefehle erlaubt.
10	Es sind nur die Befehle „Schreiben der Daten" und „Schreiben des Schlüssels und der Daten" erlaubt.
11	Es sind alle Schreibbefehle erlaubt.

Die Bits auf den Bitstellen 3 und 4 steuern die Ausführung der Suchbefehle wie folgt:

00	Alle Suchbefehle sind erlaubt.
01	Es sind nur die Befehle „Suchen Zylinder" und „Suchen Kopf" erlaubt.
10	Es ist nur der Befehl „Suchen Kopf" erlaubt.
11	Es sind keine Suchbefehle erlaubt.

Bitstelle 5 steuert die Ausführung von Diagnostikbefehlen wie folgt:

0	Diagnostik-Schreibbefehle sind nicht erlaubt
1	Diagnostik-Schreibbefehle sind erlaubt

Die Bitstelle 6 hat keine Bedeutung. Bitstelle 7 wird auf 1 gesetzt, um einen Gerätefehler zu veranlassen; damit kann Befehlswiederholung durchgeführt werden, was zur Beseitigung von Datenfehlern recht nützlich ist. In jeder Befehlskette darf der Befehl „Setzen der Dateimaske" nur ein einziges Mal auftreten. Ein nochmals gegebener Befehl dieser Art weist der Bitstelle 3 (Ungültige Folge) des Prüfbytes 1 eine 1 zu; ferner wird auch „Befehlszurückweisung" signalisiert. Die Dateimaske wird am Ende jeder Kette von Kanalbefehlswörtern wieder auf 0 gelöscht. Bei Fehlen dieses Befehles wird eine Dateimaske angenommen, die aus lauter Nullen besteht.

d) 13 —— Zurückführen (Recalibrate)
Unter der Voraussetzung, daß die Dateimaske eine solche Operation erlaubt, wird der Zugriffsmechanismus auf den Zylinder 0 und auf den Schreib/Lesekopf 0 eingestellt.

e) 04 —— Prüfen (Sense)
Es werden die sechs oder 24 Prüfbytes zum Kanal übertragen. Die letzten vier enthalten nur Diagnostik-Informationen.

f) 0F —— Kennzeichnungsfeld übergehen (Space Count)
Dieser Befehl muß entweder an den Befehl „Schreiben der Kennzeichnung, des Schlüssels und der Daten" oder an den Befehl „Löschen" gekettet werden. Ihm darf weder ein Schreib- noch ein Löschbefehl folgen; auch das Setzen der Dateimaske darf sich nicht anschließen. Dieser Befehl hilft, die Daten wiederzuentdecken, wenn das Kennzeichnungsfeld defekt ist. Ein nachfolgender geketteter Befehl „Lesen des Schlüssels und der Daten" bzw. „Lesen der Daten" liest dann den Satz mit dem defekten Kennzeichnungsfeld. Ist dieser Befehl an einen Lese-, Schreib- oder Durchsuchbefehl gekettet, so wird über das nächste Kennzeichnungsfeld hinweggegangen, und an Stelle der vom Kennzeichnungsfeld erlangten Längenangaben für Schlüsselfeld und Datenfeld werden drei Bytes vom Kanal gesendet. Das gleiche gilt für einen vorausgehenden Befehl der gleichen Art. Erfolgt eine Kettung dieses Befehls an einen anderen als die genannten, so wird über das Kennzeichnungsfeld von R_0 hinweggegangen, so daß das Schlüsselfeld und das Datenfeld von R_0 gelesen werden können.

g) 17 —— Restore
Wird von Plattenspeichergeräten wie ein Nop-Befehl angesehen.

Die Sondereinrichtung Zwei-Kanal-Schalter ermöglicht den Anschluß einer Steuereinheit an zwei Kanäle. Diese beiden Kanäle dürfen durchaus zu verschiedenen Computern gehören. Bei Vorhandensein dieser Sonderausrüstung können die beiden folgenden Befehle gegeben werden:

a) B4 —— Reservieren des Gerätes (Device Reserve)

b) 94 —— Freigeben des Gerätes (Device Release)
Durch den ersten Befehl kann eine Steuereinheit für einen bestimmten Kanal reserviert werden. Ist sie erst einmal reserviert, so kann der andere Kanal erst dann wieder zu ihr zugreifen, wenn der Freigabebefehl vom reservierten Kanal gegeben wurde.

Es kommt zur Befehlszurückweisung, wenn diesen Befehlen der Befehl „Setzen der Dateimaske" vorausgeht.

Zwei Befehle werden bei der Umdrehungspositionsbestimmung [1] benutzt.

a) 22 —— Lesen Sektor (Read Sector)

1) Im angloamerikanischen Sprachraum als „Rotational Position Sensing", abgekürzt RPS, bezeichnet.

b) 23 —— Setzen Sektor (Set Sector)
Der Befehl „Lesen Sektor" überträgt vom Gerät ein Byte an den Kanal. Dieses Byte nennt die Nummer des Sektors, wobei diese vom Indexpunkt des rotierenden Plattenstapels an gezählt werden. Der Befehl „Setzen Sektor" sendet ein Byte zur Steuereinheit. Enthält dieses den sedezimalen Wert FF, so wird der Befehl als Nop-Befehl angesehen. Verweist dieses Byte dagegen auf eine gültige Sektornummer für den betreffenden Plattenstapel, so wird er akzeptiert. Bei ungültiger Sektornummer kommt es zur Befehlszurückweisung. Bei einem Blockmultiplexkanal bewirkt dieser Befehl, daß der Versuch unternommen wird, das Gerät in dem Augenblick wieder mit dem Kanal zu verbinden, in dem der spezifizierte Sektor erreicht ist. Kann diese Verbindung deshalb nicht erfolgen, weil der Kanal in Arbeit ist, so wird der Versuch bei jeder Plattenumdrehung wiederholt. Einige Plattenspeichergeräte besitzen diese Einrichtung nicht. Bei diesen Geräten kommt es deshalb zur Befehlszurückweisung, wenn diese Befehle in einem Kanalprogramm auftauchen.

6.18. Das Gerätestatusbyte und die Prüfbytes bei Plattenspeichergeräten

Die einzelnen Bitstellen des Gerätestatusbytes haben folgende Bedeutungen:

a) Bitstelle 0 hat keine Bedeutung.

b) Bitstelle 1 —— Statusänderung
Wird die Bitstelle gleichzeitig mit der Bitstelle 3 (In Arbeit) eine 1 zugewiesen, so bedeutet das, daß die Steuereinheit beschäftigt ist. Wird diese Bitstelle gleichzeitig mit der Bitstelle 5 (Geräteende) auf 1 gesetzt, so wird damit ein erfolgreiches Durchsuchen angezeigt.

c) Bitstelle 2 —— Steuereinheitenende
Dieser Bitstelle wird eine 1 zugewiesen, wenn die Steuereinheit nicht mehr in Arbeit ist. Es wäre noch zu bemerken, daß die Befehlskettung beendet wird, wenn Steuereinheitenende festgestellt wurde.

d) Bitstelle 3 —— In Arbeit
Durch eine 1 auf dieser Bitstelle zusammen mit einer 1 auf der Bitstelle 1 wird angezeigt, daß die Steuereinheit beschäftigt ist; fehlt die 1 auf der Bitstelle 1, so ist das Gerät in Arbeit. Alle Datenübertragungen sowie alle Befehle „Durchsuchen" erfordern bei ihrer Ausführung die Mitwirkung der Steuereinheit, Suchbefehle dagegen nicht. Wenn somit dieser Bitstelle eine 1 zugewiesen ist, ohne daß auch das Bit 1 gleich 1 ist, so heißt das letzten Endes, daß eine vorangegangene Einstelloperation (Suchoperation) noch nicht beendet ist.

e) Bitstelle 4 —— Kanalende

f) Bitstelle 5 —— Geräteende
Durch eine 1 auf dieser Bitstelle wird angezeigt, daß der Zugriffsmechanismus für eine neue Operation frei ist.

g) Bitstelle 6 —— Gerätefehler
Dieser Bitstelle wird immer dann eine 1 zugewiesen, wenn irgendein Bit der Prüfbytes 0 und 1 nicht gleich 0 ist.

h) Bitstelle 7 —— Geräte-Ausnahmebedingung
Wenn bei einem der nachfolgenden Befehle eine Datenlänge von 0 im Kennzeichnungsfeld entdeckt wird, so wird diese Bitstelle auf 1 gesetzt:

(1) Lesen des Spurbeschreibungssatzes
(2) Lesen des Schlüssels und der Daten
(3) Lesen der Kennzeichnung des Schlüssels und der Daten
(4) Lesen der Daten
(5) Lesen der Anfangsspur
(6) Schreiben des Schlüssels und der Daten
(7) Schreiben der Daten

Für das Datenfeld erfolgt kein Datentransport; der Inhalt des Schlüsselfeldes wird jedoch, falls erforderlich, übertragen.

Die in den Prüfbytes enthaltenen Informationen sind für den Programmierer nur dann von grundsätzlichem Interesse, wenn sie in den ersten beiden Prüfbytes stehen. Den meisten anderen Informationen kommt hauptsächlich bei Wartung und Instandsetzung eine bestimmte Bedeutung zu. Die wesentlichen Prüfinformationen sind die folgenden:

1. Prüfbyte 0

a) Bitstelle 0 — Befehlszurückweisung
Befehlszurückweisung erfolgt in den folgenden Fällen:
- Ungültiger Befehlsschlüssel
- Ungültige Befehlsfolge
- Ungültige Suchadresse
- Verletzung der Dateimaske bei Schreibbefehlen

Bei einigen Geräten kommt es auch zur Befehlszurückweisung, wenn auf einer Spur die Hausadresse vermißt wird. Andere wiederum weisen Befehle dann zurück, wenn das Kennzeichnungsfeld von R_0 bei einer defekten Spur auf sich selbst verweist oder wenn Schreibbefehle vorliegen, obwohl der Schalter für die Schreibsperre bei dem betreffenden Laufwerk eingeschaltet ist. Auch kommt es zur Befehlszurückweisung, wenn ein Steuerungsbefehl mit ungültigem oder unvollständigem Argument gegeben wird.

b) Bitstelle 1 — Eingriff erforderlich
Das Laufwerk ist nicht betriebsbereit oder nicht mit dem System verbunden.

c) Bitstelle 2 — Fehler in der Ausgangsleitung
Ein Paritätsfehler bei Daten oder bei Befehlen wird in dieser Bitstelle festgehalten.

d) Bitstelle 3 — Ausrüstungsfehler
Nicht alle Bitstellen des Prüfbytes 2 enthalten Nullen, d.h. es liegt ein Hardwarefehler im Laufwerk oder in der Steuereinheit vor.

e) Bitstelle 4 — Datenfehler
Durch die Steuereinheit wurden unsaubere Daten vom Gerät empfangen. Bei einigen Datenfehlerarten können Wiederholungen von der Steuereinheit gestartet werden.

f) Bitstelle 5 — Datenverlust
Zu einem Datenverlust kann es innerhalb einer Befehlskette bei jedem Befehl kommen. Er kann aber auch bei jeder durch Lese- oder Schreibbefehle veranlaßten Datenübertragung eintreten. Beim Schreiben wird der Rest des Satzes mit Nullen aufgefüllt. Beim Lesen wird der Rest der Daten noch bis zur Steuereinheit übertragen, er gelangt jedoch nicht mehr in den Hauptspeicher. Einige Steuereinheiten leiten von sich aus Wiederholungsversuche ein. Sind diese erfolgreich, kommt es zu keiner Fehlermeldung.

g) Bitstelle 6 — Spurbedingung (defekte Spur)
Ein MS-Befehl geht zu einer defekten Spur oder einer Ersatzspur über bzw. kommt von dort. Eine Einzelspuroperation, die nicht gleich „Durchsuchen Spuradresse", „Lesen

Spuradresse" oder „Lesen Spurbeschreibungssatz" ist, greift zu einer defekten Spur zu. Bei einigen Plattenspeichergeräten wird diese Bitstelle nicht benutzt.

h) Bitstelle 7 — Suchfehler (Einstellfehler)
Einige Plattenspeichergeräte benutzen diese Bitstelle überhaupt nicht. Bei einigen wird ihr eine 1 zugewiesen, wenn bei einer Suchoperation der Zugriffsmechanismus nicht korrekt eingestellt wird. Bei einigen Geräten wird diese Bitstelle auch gesetzt, wenn eine ungültige Suchadresse vorliegt oder wenn bei einem Suchbefehl weniger als sechs Bytes übertragen werden.

2. Prüfbyte 1

a) Bitstelle 0 — Fehler im Kennzeichnungsfeld
Das Bit 4 des Prüfbytes 0 ist ebenfalls gleich 1. Es wurde ein Datenfehler im Kennzeichnungsfeld entdeckt. Es wäre noch zu bemerken, daß der Datenfehler im Schlüsselfeld oder im Datenfeld vorliegt, wenn dieses Bit gleich 0 ist, das Bit 4 des Prüfbytes 0 dagegen gleich 1.

b) Bitstelle 1 — Spurenüberlauf
Bevor das Satzende geschrieben werden konnte, wurde der Indexpunkt entdeckt. Bei den folgenden beiden Befehlen kann diese Bedingung gesetzt werden:
(1) Schreiben des Spurbeschreibungssatzes
(2) Schreiben der Kennzeichnung, des Schlüssels und der Daten

c) Bitstelle 2 — Zylinderende
Es wurde das Endes des Zylinders festgestellt, bevor die Befehlskette abgeschlossen werden konnte.

d) Bitstelle 3 — Ungültige Folge
Neben dieser Bitstelle wird auch der Bitstelle für die Befehlszurückweisung eine 1 zugewiesen. Diese Bedingung tritt dann auf, wenn entweder zwei Befehle „Setzen Dateimaske" in einer Befehlskette vorhanden sind oder wenn eine ungültige Folge von Kanalbefehlswörtern vorliegt.

e) Bitstelle 4 — Kein Satz gefunden
Während des Ablaufs der Befehlskettung wurde zweimal der Indexpunkt überschritten, ohne daß zwischendurch ein Lese- oder Schreibbefehl für Daten oder für die Spuradresse oder für den Spurbeschreibungssatz gegeben wurde. Dieser Bitstelle wird auch dann eine 1 zugewiesen, wenn sich keine Daten auf der Spur befinden; in diesem Fall enthält auch die Bitstelle 6 dieses Bytes eine 1. Bei Mehrspuroperationen unterbleibt grundsätzlich das Setzen dieser Bitstelle.

f) Bitstelle 5 — Dateischutz
Ein Such- oder ein Schreibbefehl verletzt die Dateimaske. Bei einem Schreibbefehl wird die Bitstelle für die Befehlszurückweisung ebenfalls auf 1 gesetzt.

g) Bitstelle 6 — Fehlende Adreßmarke
Auf der Spur sind zwei aufeinanderfolgende Sätze mit dem gleichen Bit 0 in der Adreßmarke aufgezeichnet. Das bedeutet aber, daß ein Satz auf der Spur fehlt. Dieser Bitstelle wird auch dann eine 1 zugewiesen, wenn zwei Indexpunkte nacheinander den Schreib/Lese-Kopf passieren, ohne daß sich zwischen sie eine Adreßmarke geschoben hat.

h) Bitstelle 7 — Unvollständiger Überlauf
Diese Bitstelle hat nur bei der Überlaufeinrichtung (Spurenverbindung) eine Bedeutung. Sie wird in den folgenden Fällen auf 1 gesetzt:

(1) zusammen mit der Bitstelle 6 des Prüfbytes 0, wenn ein Überlauf zu oder von einer defekten Spur eingetreten ist;
(2) zusammen mit der Bitstelle 7 des Prüfbytes 0, wenn der Überlauf zu einer falschen Spur erfolgt (der Bitstelle 7 kann nur beim Schreiben eine 1 zugewiesen werden);
(3) zusammen mit der Bitstelle 0 des Prüfbytes 0 und der Bitstelle 5 des Prüfbytes 1, wenn der Überlauf die Grenze einer geschützten Datei überschreitet.

Die hier nicht beschriebenen Bitstellen der restlichen Prüfbytes sind ohne Interesse für den Programmierer, sie enthalten Diagnostik-Informationen, die für die Behebung von Fehlern an den Geräten von Wichtigkeit sind.

6.19. Überblick über die Speichereinheiten mit direkter Zugriffsmöglichkeit (SEDZ)

Im Anhang A sind die verschiedensten Angaben, wie Spurkapazität, Zahl der Spuren pro Zylinder, mittlere Zugriffszeit usw. über die am häufigsten eingesetzten Speichereinheiten mit direkter Zugriffsmöglichkeit (SEDZ) zusammengetragen. Der Begriff „SEDZ" bezieht sich auf jedes EA-Gerät, bei dem die Zugriffszeit zu einem Satz relativ unabhängig von dem Platz ist, auf dem der Satz im Datenträger untergebracht ist. Er gilt also sowohl für Magnetplattenspeichergeräte als auch für Magnettrommelspeichergeräte. Bei einer Trommel ist eine Oxydschicht auf der Oberfläche eines langen zylindrischen Körpers aufgebracht. Eine Serie von Schreib-/Lese-Köpfen bestreicht Spuren auf der Trommeloberfläche, von ihr nur durch eine dünne Luftschicht getrennt. In den Spuren werden die Daten wie bei den Magnetplatten aufgezeichnet. Die Schreib-/Leseköpfe sind fest montiert und sind damit einer Spur fest zugeordnet. Typisch bei Trommelspeichern sind ihre höheren Kosten gegenüber Plattenspeichern und ihre geringere Speicherkapazität. Dafür sind sie schneller als Plattenspeicher.

Der Plattenstapel IBM 1316 mit 7,25 Mill. Bytes Kapazität wird in ein Plattenspeichergerät des Typs IBM 2311 eingesetzt. Eins bis sechzehn dieser Geräte werden von einem Steuergerät IBM 2841 gesteuert; dieses ist auf sechs Prüfbytes ausgerichtet. Der Plattenstapel IBM 2316 mit einer Kapazität von 29 Millionen Bytes gehört zu den Gerätetypen IBM 2314 und IBM 2319. Hinter den Typenbezeichnungen IBM 2312, IBM 2313, IBM 2314, IBM 2318 und IBM 2319 verbirgt sich praktisch das gleiche Gerät; sie unterscheiden sich nur in der Zahl der Laufwerke pro Gerät. Zwischen 1 und 8 Laufwerke können von einem Steuergerät betreut werden. Das Steuergerät ist entweder eine IBM 2314 oder ein integrierter Dateiadapter [1], eine Kombination von Selektorkanal und Steuergerät. Eine solche Anschlußmöglichkeit ist bei verschiedenen Modellen der IBM /370 verfügbar. Sechs Prüfbytes sind vorgesehen. Einige Modelle der IBM 2314 werden mit einem Steuergerät IBM 2844 verbunden. Bei einer solchen Verbindung können zwei getrennte unabhängige Operationen zur gleichen Zeit auf verschiedenen Geräten erfolgen.

Die Umdrehungspositionsbestimmung (RPS) steht für die Typen IBM 2311 und IBM 2314 nicht zur Verfügung. Bei einigen Modellen der IBM 2311 fehlt auch der Befehl „Spezielles Schreiben der Kennzeichnung, des Schlüssels und der Daten" (Befehlsschlüssel 01).

Der Plattenstapel IBM 3336 wird entweder auf die Magnetplattengeräte des Typs IBM 3330 oder des Typs IBM 3333 montiert. Die Laufwerke können entweder an ein Steuergerät

1) Im angloamerikanischen Sprachraum als „Integrated File Adapter", abgekürzt IFA bezeichnet.

des Typs IBM 3830 oder über einen integrierten Dateiadapter oder eine integrierte Speichersteuerung [1] direkt an die meisten Modelle des Systems IBM /370 angeschlossen werden. Unter einer integrierten Speichersteuerung versteht man eine Kombination zwischen einem Multiplexkanal im Einpunktbetrieb und einem Steuergerät. Das Modell 1 des Plattenstapels IBM 3336 faßt rd. 100 Mill. Bytes, das Modell 11 das Doppelte, also rd. 200 Mill. Bytes. Bis zu 32 Laufwerke können mit einem Steuergerät verbunden werden. Erfolgt der Anschluß über einen Multiplexkanal, so wird normalerweise für die IBM 3330 die RPS-Einrichtung benutzt. Das Steuergerät ist so entworfen worden, daß es bis zu acht getrennte Operationen zur gleichen Zeit auf verschiedene Laufwerke oder das gleiche Laufwerk beziehen kann. Bie einem Multiplexkanal im Einpunktbetrieb heißt das, daß die Operationen in der Reihenfolge vervollständigt werden, in der die Daten verfügbar sind und nicht in der Reihenfolge, in der die Befehle gegeben wurden. Es werden 24 Prüfbytes angelegt.

Die Magnetplattenspeichergeräte IBM 2305 werden über ein Steuergerät des Typs IBM 2835 angeschlossen. Es enthält sechs fest auf dem Laufwerk montierte Platten. Ein separater Schreib/Lese-Kopf ist für jede Spur jedes Zylinders vorgesehen. Dadurch werden die Operationen mit hoher Geschwindigkeit ausgeführt. Das Modell 1 besitzt eine Kapazität von 5,4 Mill. Bytes, das Modell 2 eine solche von 11,25 Mill. Bytes. Diese Geräte können nur an einen Multiplexkanal im Einpunktbetrieb angeschlossen werden. Acht Operationen können gleichzeitig ablaufen. Es sind 24 Prüfbytes vorgesehen. Die RPS-Einrichtung wird benutzt.

Das Magnetplattenspeichergerät IBM 3340 wird über den gleichen ISC-Direktanschluß oder über ein Steuergerät des Typs IBM 3830 mit dem System IBM /370 verbunden wie die IBM 3330. Für die IBM 3340 gibt es keine Plattenstapel, sondern sogenannte Datenmoduln mit der Typbezeichnung IBM 3348. Diese bestehen aus sechs Platten, wobei die Schreib/Lese-Köpfe in die Moduln eingebaut sind. Der Modul IBM 3348, Modell 35 besitzt eine Kapazität von 35 Millionen Bytes; bei den Modellen 70 und 70F ist die Kapazität auf 70 Mill. Bytes gesteigert. Die Umdrehungspositionsbestimmung ist auf Wunsch verfügbar. Eine Befehlswiederholung entfällt bei der IBM 3340. Es werden 24 Prüfbytes benutzt. Ersatzspuren sind nicht erforderlich, da die IBM 3340 die Eigenschaft besitzt, defekte Spurenteile in ähnlicher Weise zu überspringen, wie es bei Magnetbändern geschieht. Ein Befehl ist bei der IBM 3340 anders gestaltet: der Befehl „Schreiben der Spuradresse" veranlaßt die Aufzeichnung von sieben statt fünf Bytes. Die ersten beiden Bytes zeigen Informationen über Oberflächendefekte an. Diese Informationen werden werksseitig aufgebracht und brauchen normalerweise nicht geändert zu werden. Die IBM 3348, Modell 70F, besitzt eine Anzahl fester Schreib/Lese-Köpfe, wie das bei der IBM 2305 der Fall ist. Sie decken gerade die ersten fünf Zylinder ab. Der Rest des Moduls gleicht dem Modul des Modells 70. Die Benutzung der fest montierten Köpfe sichert schnelle Zugriffe zu den erwähnten fünf Anfangszylindern.

Das Magnettrommelspeichergerät IBM 2301 nimmt über 4 Mill. Zeichen (200 Spuren zu je 20483 Bytes) auf und wird an ein Steuergerät des Typs IBM 2841 angeschlossen. Jeweils sechs Prüfbytes sind für jedes Laufwerk verfügbar. Die RPS-Einrichtung ist nicht vorhanden. Das Trommelspeichergerät IBM 2303 wird über das Steuergerät IBM 2841 an einen Kanal angeschlossen und arbeitet wie ein Plattenspeichergerät. Da jedoch seine Köpfe fest montiert sind und folglich keine Suchbewegung durchzuführen brauchen, erfordert der Zugriff zu den aufgezeichneten Sätzen nur durchschnittlich eine Zeit von 8,6 ms, bedingt durch die Verzögerung infolge der Trommelumdrehungen. Die IBM 2303 wird so programmiert, als ob sie nur aus einem Zylinder bestehen würde (Zylinder 0).

1) Im angloamerikanischen Sprachraum als „Integrated Storage Control", abgekürzt ISC, bezeichnet.

Die Informationen können bei der IBM 2303 in 800 Spuren mit einer Kapazität von je 4892 Bytes aufgezeichnet werden.

Die beiden genannten Magnettrommelspeichergeräte sind die einzigen ihrer Art, die für die IBM /370 verfügbar sind. Beide sind nur in geringem Umfang in der Praxis eingesetzt.

Aufgabengruppe 14

1) Schreibe ein Kanalprogramm, durch das ein achtzigstelliger Satz auf ein Magnetband geschrieben werden kann. Der aufzuzeichnende Satz ist auf den Hauptspeicherplätzen ab 023000 gespeichert. Das Magnetband soll auf den Ladepunkt eingestellt sein.

2) Die Aufgabe 1) ist wie folgt abzuändern: Das Magnetband ist zunächst zurückzuspulen. Nach dem Aufzeichnen des Satzes ist anschließend eine Bandmarke zu schreiben.

3) Schreibe ein Supervisorprogramm, mit dessen Hilfe die Spuradreßsätze auf einen Plattenstapel IBM 2316 geschrieben werden können.

   Alle Spuren sollen dabei als Primärspuren gelten; die Möglichkeit, daß einige Spuren schadhaft sein könnten, soll vernachlässigt werden.

4) Schreibe ein Kanalprogramm, durch das ein 24-stelliger Satz, der auf den Hauptspeicherplätzen ab 033 000 gespeichert ist, auf einen Plattenspeicher in den Satzbereich R_1 der Spur 1 des Zylinders 0 geschrieben wird. Es kann vorausgesetzt werden, daß die Spuradressen sowie der Spurbeschreibungssatz R_0 bereits auf dem Plattenstapel aufgezeichnet sind.

5) Modifiziere die Aufgabe 4) wie folgt: Es ist zusätzlich ein 2000 Bytes langer Satz in den Satzbereich R_2 zu schreiben; danach ist ein Dateiendesatz (EOF) nach R_3 zu schreiben.

6) Auf einen Plattenspeicher sollen Sätze fester Länge ohne Schlüssel geschrieben werden. Wieviele Sätze passen auf eine Spur einer IBM 3330, wenn die Satzlänge

   a) 80
   b) 200
   c) 10

   Bytes beträgt? Verwende dazu den Anhang H!

7) Löse die gleiche Aufgabenstellung wie bei 6) für die IBM 2314.

8) Löse die gleiche Aufgabenstellung wie bei 6) für die IBM 3340.

9) Es ist ein Kartenpaket einzulesen. Der Inhalt jeder Karte soll als 80stelliger Satz auf einen Plattenspeicher übertragen werden. Der erste Satz soll in den Satzbereich R_1 der Spur 0 des Zylinders 107 einer IBM 3330 gespeichert werden. Es existieren höchstens 2000 Karten. Hinter den letzten Datensatz ist außerdem ein Dateiendesatz (EOF-Record) zu schreiben. Das Kartenpaket ist über eine IBM 2501 mit der Adresse 014 einzulesen!

10) Die Aufgabe 9) ist wie folgt zu modifizieren: Der Inhalt der Kartenspalten 2 bis 7 ist als Schlüsselfeld zu verwenden. Die Datensätze sollen jedoch nach wie vor 80 Bytes lang sein, d.h. das Schlüsselfeld soll nach wie vor auch Aufnahme im Satz selbst finden.

11) Schreibe ein Programm, mit dessen Hilfe die Sätze aufgesucht werden können, die mit dem Programm von Aufgabe 10) auf einen Plattenspeicher geschrieben wurden. Das Kennzeichenfeld des Satzes mit dem niedrigsten Schlüssel soll auf die Speicherplätze mit den Adressen 034 000 bis 034 004 eingelesen werden. Der Inhalt des Schlüsselfeldes soll dabei als Dualzahl angesehen werden.

12) Die Aufgabe 11) ist wie folgt abzuändern: Wenn ein Satz mit dem Schlüssel DA G370 in der Datei vorhanden ist, so soll dessen Kennzeichnungsfeld ausgelesen werden. Fehlt ein solcher Datz in der Datei, so sind die Hauptspeicherstellen ab 034 000 auf Nullen zu setzen.

7. Weitere Instruktionen

7.1. Zahlen in Gleitpunktdarstellung

Die Gleitpunktdarstellung [1] von Zahlen besitzt bei der IBM /370 folgendes Aussehen:

Inhalt	±	Charakteristik							Mantisse				
Bitstelle	0	1	2	3	4	5	6	7	8	...	28 oder 60	29 oder 61	30 oder 62	31 oder 63

Abb. 7.1. Gleitpunktdarstellung von Zahlen

Die am weitesten links gelegene Bitstelle enthält das Vorzeichen, 1 bei negativen Zahlen, 0 bei positiven Zahlen. Die nächsten sieben Bitstellen enthalten die Charakteristik. Im Rest des Wortes bzw. des Doppelwortes ist die Mantisse untergebracht. Umfaßt die Mantisse 14 Sedezimalstellen, so sprechen wir von Gleitpunktzahlen doppelter Genauigkeit; umfaßt die Mantisse nur sechs Sedezimalstellen, so sprechen wir von Gleitpunktzahlen einfacher Genauigkeit.

Die Gleitpunktdarstellung besitzt die Form ± cc mmmmmm; damit ist die Zahl ± 0, mmmmmm · 16^{cc-64} gemeint, d.h. das Produkt aus einem Bruch, dessen Komma unmittelbar vor der ersten Ziffernstelle angenommen wird, und aus einer Potenz von 16, deren Exponent gleich der um 64 verminderten Charakteristik ist. Durch die Gleitpunktdarstellung können wir Zahlen erfassen, deren Absolutbeträge im Wertebereich von 00 100000 $\hat{=}$ 0,540 · 10^{-78} bis 7F FFFFFF FFFFFFFF $\hat{=}$ 0,724 · 10^{76} liegen.

Für die Gleitpunktdarstellung wollen wir einige Beispiele geben (Abb. 7.2.).

Dezimalzahl	Sedezimalzahl	Gleitpunktdarstellung
1	00000001	41 100000
10	0000000A	41 A00000
0,1	---	40 100000 9999999A
0,01	---	3F 28F5C2 8F5C28F6
0	00000000	00 000000 00000000
974	000003CE	43 3CE000
67 345	00010711	45 107110
66 371	00010343	45 103430
-67 345	FFFEF8EF	C5 107110
2 147 483 647	7FFFFFFF	48 7FFFFF FF000000
0,000 915 527 343 75	---	3E 3C0000
-0,000 915 527 343 75	---	BE 3C0000
67,345	---	42 435841 EB841EB8

Abb. 7.2. Beispiele für die Gleitpunktdarstellung von Zahlen

1) Mitunter auch als Gleitkommadarstellung bezeichnet.

Man beachte, daß die Gleitpunktdarstellung nicht vom Zweierkomplement Gebrauch macht. Gleitpunktzahlen einfacher Genauigkeit sparen Hauptspeicherplatz, da sie nur ein Wort gegenüber zwei Wörtern bei Gleitpunktzahlen doppelter Genauigkeit belegen. Selbstverständlich liefern Gleitpunktzahlen doppelter Genauigkeit bei Berechungen exaktere Resultate. Hinsichtlich der Operationsgeschwindigkeiten wäre zu sagen, daß die Operationen mit Gleitpunktzahlen einfacher Genauigkeit stets ein wenig schneller ablaufen als die gleichen Operationen mit Gleitpunktzahlen doppelter Genauigkeit.

Liegt die Mantisse zwischen $\frac{1}{16}$ und 1, d.h. ist die erste Mantissenstelle [1] nicht mit einer Null belegt, so sprechen wir von n o r m a l i s i e r t e n Gleitpunktzahlen. Bei der IBM /370 ist es nicht notwendig, daß die Gleitpunktzahlen in normalisierter Form vorliegen. Die Verwendung von u n n o r m a l i s i e r t e n Gleitpunktzahlen führt jedoch zu weniger gültigen Ziffern bei den Resultaten von Rechenoperationen.

7.2. Instruktionen für Operationen mit Zahlen in der Gleitpunktdarstellung (Gleitpunktinstruktionen)

Alle Gleitpunktinstruktionen besitzen entweder das RR-Format oder das RX-Format. Jeder dieser Instruktionen erscheint in zwei Formen; die eine Form gilt für die Operationen mit Operanden einfacher Genauigkeit, die andere für Operationen mit Operanden doppelter Genauigkeit. Die Gleitpunktinstruktionen sind standardmäßig nur in die größeren Modelle der Computerfamilie IBM /370 eingebaut, bei den kleineren Modellen müssen sie gesondert bestellt werden.

Von den Gleitpunktinstruktionen werden Daten in den 16 allgemeinen Registern nicht angesprochen. Stattdessen beziehen sie sich immer auf eines oder auf zwei von vier Gleitpunktregistern, die jeweils ein Doppelwort umfassen. Jedes dieser Register kann somit eine Gleitpunktzahl doppelter Genauigkeit aufnehmen. Die Gleitpunktregister sind mit 0, 2, 4 und 6 numeriert. Die in diesem Abschnitt aufgeführten 44 Instruktionen sowie die 7 Instruktionen, die im Abschnitt 7.3. beschrieben sind, sind die einzigen Instruktionen, die von diesen speziellen Registern Gebrauch machen.

Die Tabelle in Abb. 7.3. enthält eine Übersicht über die ersten 44 in diesem Abschnitt zu behandelnden Gleitpunktinstruktionen.

Ist bei den Instruktionen in der letzten Spalte ein B aufgeführt, so heißt das, daß die betreffenden Instruktionen den Bedingungsschlüssel neu setzen. Dieser wird genau so gesetzt wie bei den bereits besprochenen ähnlichen Instruktionen für duale Festpunktdaten bzw. für gepackte Dezimaldaten, die die gleiche Operation bewirken, d.h. auf 0, 1 oder 2, je nachdem, ob das Resultat gleich Null, negativ oder positiv ist. Gleitpunktinstruktionen führen nie auf den Wert 3 beim Bedingungsschlüssel. Die Ausführung der in Abb. 7.3 zusammengestellten Gleitpunktinstruktionen erfolgt grundsätzlich in der gleichen Weise wie für die bereits besprochenen Instruktionen mit den Daten in der Festpunktdarstellung (Abschnitte 4.3. und 4.4.), nur daß eben in die Arithmetik Zahlen in Gleitpunktdarstellung eingehen.

Die Instruktionen für Zahlen mit doppelter Genauigkeit führen für jeden Berechnungsschritt 14 Sedezimalziffern mit; im Bedarfsfall werden nach rechts Nullen angehängt. Ergebnisse werden stets nach 14 Stellen abgeschnitten. Infolge dieses Abschneidens und infolge von Rundungen kann es gelegentlich zu Resultaten kommen, die nur bis auf 12 gültige Sedezimalziffern exakt sind. Die Instruktionen für Zahlen mit einfacher Genauigkeit zerstören weder den Inhalt der rechten Hälfte der Gleitpunktregister noch

1) Das bedeutet, daß die erste Sedezimalziffer der Mantisse, die Sedezimalziffer höchster Ordnung also, von Null verschieden ist.

1) Instruktionen im RR-Format

Instruktionen		Genauigkeit		
Deutsche Bezeichnung	Englische Bezeichnung	einf.	dopp.	
Laden	Load	38 LER	28 LDR	
Vergleichen Register	Compare Register	39 CER	29 CDR	B
Addieren normalisiert Reg.	Add normalized Register	3A AER	2A ADR	B
Subtrahieren normal. Reg.	Subtract normalized Reg.	3B SER	2B SDR	B
Multiplizieren Register	Multiply Register	3C MER	2C MDR	
Dividieren Register	Divide Register	3D DER	2D DDR	
Addieren unnormalisiert R.	Add unnormalized Reg.	3E AUR	2E AWR	B
Subtrahieren unnormal. Reg.	Subtract unnorm. Reg.	3F SUR	2F SWR	B
Umladen positiv	Load Positive	30 LPER	20 LPDR	B
Umladen negativ	Load Negative	31 LNER	21 LNDR	B
Umladen und Prüfen	Load and Test	32 LTER	22 LTDR	B
Umladen mit Vorzeichenumkehrung	Load Complement	33 LCER	23 LCDR	B
Halbieren Register	Halve Register	34 HER	24 HDR	

2) Instruktionen im RX-Format

Instruktionen		Genauigkeit		
Deutsche Bezeichnung	Englische Bezeichnung	einf.	dopp.	
Laden	Load	78 LE	68 LD	
Vergleichen	Compare	79 CE	69 CD	B
Addieren normalisiert	Add normalized	7A AE	6A AD	B
Subtrahieren normalisiert	Subtract normalized	7B SE	6B SD	B
Multiplizieren	Multiply	7C ME	6C MD	
Dividieren	Divide	7D DE	6D DD	
Addieren unnormalisiert	Add unnormalized	7E AU	6E AW	B
Subtrahieren unnormalisiert	Subtract unnormalized	7F SU	6F SW	B
Speichern	Store	70 STE	60 STD	

Abb. 7.3. Gleitpunktinstruktionen

ändern sie ihn in irgendeiner Weise ab. Bei allen Rechenschritten dieser Instruktionen werden 7 Sedezimalziffern mitgeführt; im Bedarfsfalle werden nach rechts Nullen ergänzt. Ergebnisse werden jedoch grundsätzlich nach sechs Sedezimalziffern abgeschnitten.

Nachstehend folgen einige Bemerkungen zu den einzelnen Instruktionen:

1) Laden
Der zweite Operand wird in das durch den ersten Operanden spezifizierte Register übertragen.

2) Vergleichen
Der zweite Operand wird vom ersten Operanden subtrahiert; das Ergebnis wird jedoch nicht festgehalten, sondern dient lediglich dazu, den Bedingungsschlüssel zu setzen. Sind die beiden Operanden gleich, so ist der BS [1] gleich 0, ist der erste Operand kleiner als der zweite, so wird der BS auf 1 gesetzt, und ist schließlich der zweite Operand kleiner als der erste, so wird der BS auf den Wert 2 gesetzt.

Man beachte, daß der Vergleich der beiden Zahlen in Gleitpunktdarstellung

45 067440 und
44 674400

zum Vergleichsergebnis „gleich" führt, da durch die beiden Darstellungen die gleiche Zahl festgelegt ist. Der BS wird also auf den Wert 0 gesetzt.

3) Addieren normalisiert
Der zweite Operand wird auf den ersten Operanden addiert; die Summe ersetzt den ersten Operanden. Der BS wird wie folgt gesetzt: a) auf 0, wenn die Summe gleich 0 ist, b) auf 1, wenn die Summe negativ ist und c) auf 2, wenn die Summe positiv ist. Addieren wir beispielsweise

44 674400 89654000 auf 44 12809E 79001000

mit einer AE-Instruktion, so ergibt sich als Resultat

44 79C49E 79001000,

da die rechte Hälfte des Registers nicht benutzt wird. Verwenden wir dagegen eine AD-Instruktion, so ergibt sich als Resultat

44 79C49F 02655000.

4) Subtrahieren normalisiert
Der zweite Operand wird vom ersten Operanden subtrahiert und die Differenz ersetzt den ersten Operanden. Der BS wird wie beim Addieren gesetzt.

5) Multiplizieren
Die beiden Operanden werden miteinander multipliziert; das Produkt ersetzt den ersten Operanden. Das Resultat wird abgeschnitten, damit es in ein Wort bzw. in ein Doppelwort hineinpaßt.

6) Dividieren
Der erste Operand wird durch den zweiten Operanden dividiert. Der Quotient ersetzt den ersten Operanden, der Rest geht verloren.

7) Addieren unnormalisiert bzw. Subtrahieren unnormalisiert
Alle arithmetischen Instruktionen mit Ausnahme dieser acht Instruktionen (AUR, AWR, AU, AW, SUR, SWR, SU und SW) ergeben immer ein normalisiertes Resultat, selbst wenn die Operanden in unnormalisierter Form vorliegen; bei diesen acht Instruktionen ist das nicht der Fall.

Bei der Addition oder Subtraktion von Gleitpunktzahlen wird zunächst die Mantisse der Zahl mit der kleineren Charakteristik bei gleichzeitiger Erhöhung der Charakteristik um 1 solange um eine Stelle nach rechts verschoben, bis die Charakteristiken der beiden Zahlen gleich sind. Erst dann werden die Mantissen der beiden Zahlen addiert bzw. subtrahiert. Man verfolge ein erstes Beispiel:

1) BS ist die Abkürzung von Bedingungsschlüssel.

```
  44  674400  89654000
+ 42  123001  11348997
```

Zunächst wird die Mantisse der zweiten Zahl um 2 Sedezimalstellen nach rechts verschoben. Nach dieser Verschiebung ergibt sich:

```
  44  674400  89654000
+ 44  001230  01113489
  44  675630  8A767489
```

Betrachten wir nun noch ein zweites Beispiel:

```
  44  674400  89654000
+ C4  5F0000  00001000
  44  084400  89653000
```

Hier ergibt sich ein Resultat, das unnormalisiert ist, da die erste Mantissenstelle nach der Addition eine 0 enthält.
Abgesehen von diesen acht Instruktionen, die das Ergebnis nicht normalisieren, normalisieren alle anderen Instruktionen das Resultat. Somit ergibt sich bei den entsprechenden Additionsinstruktionen als Summe

```
  43  844008  96530000
```

8) Speichern
Die im Register stehende Gleitpunktzahl wird gespeichert.

9) Umladen positiv
Der absolute Wert des zweiten Operanden wird in das durch den ersten Operanden spezifizierte Register übertragen. Der BS kann nur auf 0 oder 2 gesetzt werden.

10) Umladen negativ
Das Negative des absoluten Betrages des zweiten Operanden wird in das durch den ersten Operanden spezifizierte Register übertragen; Der BS kann nur auf 0 oder 1 gesetzt werden.

11) Umladen und Prüfen
Es läuft die gleiche Operation wie beim Laden ab, nur daß zusätzlich noch der BS gesetzt wird.

12) Umladen mit Vorzeichenumkehrung
Die im zweiten Register stehende Zahl wird mit umgekehrtem Vorzeichen ins erste Register geladen. Der BS wird auf 0, 1 oder 2 gesetzt.

13) Halbieren
Bei dieser speziellen Gleitpunktinstruktion wird der zweite Operand durch 2 dividiert und anschließend in das Register übertragen, auf das sich der erste Operand bezieht. Wenn der zweite Operand eine aus lauter Nullen bestehende Mantisse besitzt, wird das Ergebnis zu einer Gleitpunktnull umgeformt, d.h. die Charakteristik wird auf 0 gebracht.

Das folgende Beispielprogramm addiert die auf dem Speicherplatz 003 800 stehende Zahl zu der auf Speicherplatz 003 804 stehenden Zahl und stellt die Summe in das Doppelwort mit der Adresse 003 808. Dabei nehmen wir an, daß das allgemeine Register 3 den Wert 0000 3800 enthält.

Zu unserem Beispielprogramm von Abb. 7.4. wäre noch zu bemerken, daß die rechte Hälfte des Gleitpunktregisters 2 niemals benutzt wurde. Infolgedessen bleibt ihr Inhalt unzerstört.

Hauptspeicheradresse	Inhalt	
	a) Daten	
003 800	44 124889	
003 804	C3 F17333	
003 808	78 111111	
	b) Instruktionen	
003 810	LE	78 2 03 000
003 814	AE	7A 2 03 004
003 818	STE	70 2 03 008

Nach Durchführung dieser drei Instruktionen enthält
003 808 das Ergebnis 43 33155D
Das Gleitpunktregister 2 enthält den gleichen Wert
43 33155D

Abb. 7.4. Beispielprogramm

7.3. Zahlen in Gleitpunktdarstellung mit erweiterter Genauigkeit

In einige Modelle der Computerfamilie IBM /370 ist auch eine Einrichtung eingebaut, die Zahlen in Gleitpunktdarstellung mit erweiterter Genauigkeit verarbeitet. Die meisten anderen Modelle können mit dieser Einrichtung zusätzlich auf Bestellung ausgerüstet werden.

Diese Einrichtung ermöglicht die Verwendung von sieben zusätzlichen Instruktionen, die die Gleitpunktregister ansprechen. Eine Gleitpunktzahl erweiterter Genauigkeit belegt zwei Doppelwörter; in jedem Doppelwort steht dabei ein Teil der Gleitpunktzahl. Die Exponenten dieser beiden Teilzahlen unterscheiden sich um 14 (sedezimal 0E). Somit bleiben zur Aufnahme der Mantisse in beiden Registern gerade noch 28 Sedezimalstellen übrig.

Gleitpunktzahlen mit erweiterter Genauigkeit können entweder in das Registerpaar 0-2 oder in das Registerpaar 4-6 geladen werden. Die Adressierung muß sich dabei jeweils auf das Register 0 bzw. auf das Register 4 beziehen. Es kommt zu einem Fehler [1], wenn zur Adressierung ein anderes Register als 0 bzw. 4 herangezogen wird. Der wertmäßig kleinste Anteil der Gleitpunktzahlen, d. h. die Mantissenstellen niedriger Ordnung, befinden sich stets in den Registern 2 bzw. 6.

Instruktion		Operationsschlüssel		Instruktionsformat
Deutsche Bezeichnung	Englische Bezeichnung	symb.	echt	
Laden mit Runden	Load Rounded	LRDR	25	RR
Laden mit Runden	Load Rounded	LRER	35	RR
Normal. Addieren	Add Normalized	AXR	36	RR
Norm. Subtrahieren	Subtract Normalized	SXR	37	RR
Multiplizieren	Multiply	MXR	26	RR
Multiplizieren	Multiply	MXDR	27	RR
Multiplizieren	Multiply	MXD	67	RX

Abb. 7.5. Instruktionen für Operationen mit Gleitpunktzahlen erweiterter Genauigkeit

1) Später werden wir erkennen, daß es sich um einen sogenannten Spezifikationsfehler handelt.

Die beiden Instruktionen AXR und SXR setzen den Bedingungsschlüssel wie gewöhnlich auf 0, 1 oder 2. Die Instruktion AXR bewirkt, daß die Gleitpunktzahl erweiterter Genauigkeit, die durch den zweiten Operanden angesprochen wird, auf die Gleitpunktzahl erweiterter Genauigkeit addiert wird, die durch den ersten Operanden spezifiziert ist. In der gleichen Weise ruft die Instruktion SXR eine Subtraktion hervor. Bei der Instruktion MXR werden zwei Gleitpunktzahlen erweiterter Genauigkeit miteinander multipliziert. Diese drei Instruktionen laufen wie folgt ab: Die Vorzeichen und die Charakteristiken der beiden Operanden werden aus dem ersten Byte des ersten Doppelwortes, d.h. vom Zahlenanteil der höheren Ordnung, herausgeholt. Es wird ferner grundsätzlich angenommen, daß das Vorzeichen des Zahlenanteiles der niedrigeren Ordnung das gleiche ist wie das des Zahlenanteiles der höheren Ordnung und daß die entsprechende Charakteristik um 14 kleiner ist. Es erfolgt deshalb keine Prüfung des ersten Bytes der zweiten Hälfte der in die Operationen eingehenden Zahlen, d.h. der die Charakteristik enthaltenden Bytes. Das erste Byte der zweiten Zahlenhälfte des Ergebnisses wird vom Computer wie folgt geformt: Auf die Vorzeichenbitstelle setzt er das Vorzeichen der ersten Zahlenhälfte, und die Charakteristik ist gleich der um 14 verminderten Charakteristik der ersten Zahlenhälfte. Ist jedoch die Charakteristik des Zahlenanteiles der höheren Ordnung kleiner als 14, so wird 128 auf die Charakteristik der zweiten Zahlenhälfte addiert. Ein Exponentenüberlauf wird erkannt, wenn der Exponent der ersten Zahlenhälfte zu groß wird; ein Exponentenunterlauf ist ebenfalls durch die auftretende Charakteristik des ersten Zahlenanteiles bestimmt. Die 28 sedezimalen Mantissenstellen einer Null enthalten sämtlich die Ziffer 0. Es ist möglich, daß die rechte Hälfte des Resultates nicht normalisiert ist, wenn sie als alleinstehende Gleitpunktzahl doppelter Genauigkeit betrachtet wird. Das ist kein Wunder, denn die Mantissenstellen der zweiten Hälfte sind ja alle unter dem Aspekt zu sehen, daß sie 14 Stellen weiter rechts als die erste Mantissenstelle der ersten Zahlenhälfte beginnen müssen, und auf der 15. Mantissenstelle der gesamten Zahl kann nun einmal eine Null vorhanden sein.

Mit den Instruktionen MXD und MXDR können zwei doppelt genaue Gleitpunktzahlen miteinander multipliziert werden; das sich ergebende Produkt besitzt die erweiterte Genauigkeit.

Die Instruktion LRER dient dazu, eine doppelt genaue Gleitpunktzahl (14 sedezimale Mantissenstellen) so zu runden, daß sich eine Gleitpunktzahl einfacher Genauigkeit (6 sedezimale Mantissenstellen) ergibt. Die Rundung wird dadurch hervorgerufen, daß eine duale 1 auf die Bitstelle 32 des absoluten Wertes der zu rundenden Zahl im Register addiert wird. Ein Exponentenüberlauf kann hierbei eintreten.

Durch die Instruktion LRDR kann eine Gleitpunktzahl erweiterter Genauigkeit so gerundet werden, daß sich eine Gleitpunktzahl doppelter Genauigkeit ergibt. Das geschieht durch Addition einer dualen 1 im Register auf die Bitstelle 72 des absoluten Wertes der zu rundenden Zahl. Hierbei kann es zu einem Exponentenüberlauf kommen.

Aufgabengruppe 15

1) Gegeben sind folgende Inhalte:

Allgemeines Register 11	0000 2000
Gleitpunktregister 4	BE 827146 01234567
Gleitpunktregister 6	41 400000 00000000
Speicherplätze 002 000 bis 002 007	44 100000 40100000

Gesucht sind die Inhalte des Gleitpunktregisters 6 nach Durchführung der folgenden Instruktionen:

```
a) AD     6A 6B 0 000
b) SD     6B 6B 0 000
c) MD     6C 6B 0 000
d) DDR    2D 64
e) ADR    2A 66
f) SER    3B 44
   LDR    28 64
g) HDR    24 64
h) DDR    2D 66
   HDR    24 66
i) AU     7E 4B 0 004
   LDR    28 64
j) AE     7A 4B 0 004
   LDR    28 64
```

2) Unter Verwendung der Daten von Aufgabe 1) sollen ein oder zwei Instruktionen niedergeschrieben werden, die die nachstehenden Werte als Ergebnisse ins Gleitpunktregister 6 stellen:

```
a)  00 000000 00000000
b)  42 100000 00000000
c)  BE 827146 00000000
d)  C1 800000 00000000
e)  42 200000 00000000
f)  BF 104E28 C02468AC
g)  41 400827 14601234
h)  40 FF7D8E B9FEDCC0
```

3) Schreibe ein Programm zur Lösung quadratischer Gleichungen der Form $ax^2+bx+x=0$. Zur Eingabe gelangen Karten, auf denen die Koeffizienten a, b und c abgelocht sind. Zur Ausgabe sollen ebenfalls Karten gelangen, auf die a, b, c sowie die beiden Lösungen x_1 und x_2 gestanzt sind. Es soll vorausgesetzt werden, daß die eingelesenen Koeffizienten nur auf reelle Lösungen führen. Alle Zahlen sollen in Gleitpunktdarstellung eingegeben bzw. ausgegeben werden; sie belegen je 16 Kartenspalten.

4) Schreibe eine Instruktionsfolge zur Berechnung von

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

Der x-Wert wird im Gleitpunktregister 0 zur Verfügung gestellt. Das Resultat, also der Wert von e^x, soll ebenfalls ins Gleitpunktregister 0 gestellt werden. Die letzte Instruktion in der Instruktionsfolge soll eine unbedingte Verzweigung zu der im Register 14 stehenden Adresse sein, also eine BCR-Instruktion, deren drittes Halbbyte F und deren viertes Halbbyte E enthält.

5) Ändere Aufgabe 4) wie folgt ab:

a) Ist $x>0$, so soll e^x direkt nach der obigen Reihenentwicklung berechnet werden.

b) Ist $x<0$, so soll dagegen zuerst e^{-x} berechnet werden. Der Wert von e^x soll dann aus der Beziehung

$$e^x = \frac{1}{e^{-x}}$$

bestimmt werden.

7.4. Halbwort-Instruktionen

Die bisher behandelten Instruktionen für Dualzahlen in Festpunktdarstellung arbeiten mit Wörtern. Zur Einsparung von Speicherplatz ist es oft wünschenswert, daß man kurze Konstanten auch in einem Halbwort speichern kann. Zum Handhaben der Halbwörter stehen uns sechs Instruktionen im RX-Format zur Verfügung (siehe hierzu Abb. 7.6.).

Speichern Halbwort	Store Halfword	40	STH	RX
Laden Halbwort	Load Halfword	48	LH	RX
Vergleichen Halbwort	Compare Halfword	49	CH	RX
Addieren Halbwort	Add Halfword	4A	AH	RX
Subtrahieren Halbw.	Subtract Halfword	4B	SH	RX
Multiplizieren Halbw.	Multiply Halfword	4C	MH	RX

Abb. 7.6. Halbwortinstruktionen

Diese Instruktionen laufen genau so ab wie die regulären Instruktionen ST, L, C, A und S. Das im Hauptspeicher stehende Halbwort wird zunächst zu einem vollen Wort ausgedehnt, indem das linke Bit (Vorzeichen) in die zusätzlichen 16 Bitstellen übernommen wird. Die Operation läuft dann genau so ab, als wenn die im Hauptspeicher stehenden Daten von Anfang an in Vollwörtern gespeichert wären.

Die Instruktion MH unterscheidet sich von den anderen Instruktionen etwas. Sie bewirkt, daß der Inhalt des im Hauptspeicher stehenden Halbwortes mit dem Inhalt des durch den ersten Operanden spezifizierten Registers multipliziert wird. Das Produkt ersetzt den Inhalt des angegebenen Registers. In diese Operation geht nur ein Register ein; in die reguläre M-Instruktion ist bekanntlich ein Registerpaar verwickelt. Nur die letzten 32 Stellen des Produktes werden zurückgehalten; eine Überlaufanzeige erfolgt nicht.

Einige Beispiele mögen die Betrachtungen über die Halbwortinstruktionen beschließen.
Inhalte vor Ausführung der Instruktionen:

Register 3 0000 8300

Speicherplätze 008300 bis 008307 FFFD 0002 9999 9999

Instruktionen		Inhalt des Registers 4 nach Ausführung der Instruktionen
LH	48 4 0 3 000	FFFF FFFD
MH	4C 4 0 3 002	FFFF FFFA
ST	50 4 0 3 004	FFFF FFFA
SH	4B 4 0 3 006	0000 0000

7.5. Instruktionen im RS-Format[1)]

Einige Instruktionen der IBM /370 erfordern die Angabe von drei Adressen. Diese Instruktionen treten im RS-Format auf, das so aufgebaut ist, wie es die Abb. 7.7. zeigt.

1) Register zu Speicher-Operation (im Engl.: Register to Storage).

Byte	0	1		2	3
Inhalt des Bytes	Operations-schlüssel	R_1	R_3	B_2	D_2

Abb. 7.7. RS-Format von Instruktionen

Wie im RX-Format belegen auch diese Instruktionen vier Bytes. Zum Unterschied gegenüber dem RX-Format kann jedoch der zweite Operand nicht indiziert werden; die damit freiwerdende Registeradresse dient zur Angabe eines dritten Operanden, ebenfalls eines allgemeinen Registers. Wir werden später noch weiteren derartigen Instruktionen begegnen; hier wollen wir zunächst nur zwei sehr wichtige betrachten.

1) Speichern mehrfach — Store Multiple | 90 | STM | RS

Der Inhalt der allgemeinen Register R_1, $R_1+1, \cdots, R_3-1$, R_3 wird in aufeinanderfolgende Speicherplätze gespeichert, die mit der effektiven Adresse $E_2=(B_2)+D_2$ beginnen. Ist R_1 größer als R_3, so werden die Register in zyklischer Reihenfolge gespeichert (zyklische Adreßfolge), d.h. auf Register 15 folgt Register 0, danach Register 1 usw. Mit dieser Instruktion können die Inhalte von einem bis zu allen sechzehn Registern weggespeichert werden.

Die Instruktion

90 D C 0 400

z.B. speichert den Inhalt von Register 13 auf die Speicherplätze 000 400 bis 000 403, den Inhalt von Register 14 auf die Speicherplätze 000 404 bis 000 407, den Inhalt von Register 15 auf die Speicherplätze 000 408 bis 000 40B, Register 0 auf 000 40C bis 000 40F usw. Zuletzt wird schließlich der Inhalt des Registers 12 auf die Speicherplätze 000 43C bis 000 43F gespeichert. Man beachte hierbei, daß das allgemeine Register 0 nicht zur Adreßbildung herangezogen wird.

2) Laden mehrfach — Load Multiple | 98 | LM | RS

LM bewirkt das Gegenteil von STM. Diese Instruktion hat zur Folge, daß die Register von R_1 bis R_3 mit dem Inhalt der bei E_2 beginnenden Speicherplätze geladen werden. Die Instruktion

98 C D 0 500

lädt somit das Register 12 mit dem Inhalt der Speicherplätze 000 500 bis 000 503 und das Register 13 mit dem Inhalt der Speicherplätze 000 504 bis 000 507.

7.6. Instruktionen im SI-Format

Instruktionen im SI-Format sind so aufgebaut, wie es die Abb. 7.8. zeigt.

Byte	0	1	2	3
Inhalt des Bytes	Operations-schlüssel	I_2	B_1	D_1

Abb. 7.8. Instruktionen im SI-Format

Bei den Instruktionen im SI-Format [1] wird durch den ersten Operanden eine Hauptspeicheradresse angegeben (genau so wie bei den Instruktionen im SS-Format). Der zweite Operand spezifiziert weder eine Hauptspeicheradresse noch ein allgemeines Register, sondern ist Teil der Instruktion selbst. Alle bei den einzelnen Instruktionen dieses Formats unter I_2 aufgeführten Informationen stellen Direktdaten dar. Diese Direktdaten gehen in die Ausführung der Instruktion unmittelbar selbst ein; sie können also nur ein Byte lang sein. Um den Ablauf einer solchen Instruktion näher erläutern zu können, wollen wir eine erste Instruktion dieses Formats betrachten:

1)

Übertragen direkt — Move Immediate	92	MVI	SI

Das Datenbyte I_2 wird auf den Hauptspeicherplatz übertragen, dessen Adresse $E_1=(B_1)+D_1$ ist.
Die Instruktion

92 FF 0 509

z.B. ersetzt den vorhergehenden Inhalt von 000 509 durch FF. Nur ein einziges Byte des Hauptspeichers wird von dieser Instruktion berührt.

Eine andere Instruktion im SI-Format ist die folgende:

2)

Prüfen mit Maske — Test under Mask	91	TM	SI

TM untersucht I_2 und das durch die Hauptspeicheradresse E_1 angesprochene Datenbyte und setzt den Bedingungsschlüssel auf folgende Werte:
Entspricht jedes Einerbit von I_2 einem in der gleichen Bitstelle des Bytes mit der Adresse E_1 stehenden Bit 0, so erhält der BS[2] den Wert 0; entspricht jedes Einerbit von I_2 einem Einerbit in der gleichen Bitstelle des Bytes auf E_1, so wird der BS auf 3 gesetzt. In allen anderen Fällen erhält der BS den Wert 1 zugewiesen. Ist I_2 gleich 00, dann wird, und das ist trivial, der BS auf 0 gesetzt. Das auf dem Hauptspeicherplatz mit der Adresse E_1 stehende Zeichen wird durch diese Operation nicht verändert. Man betrachte einige Beispiele:

a) I_2	0000 1111	BS gleich 1
Hauptspeicher	0101 0101	
b) I_2	0000 1111	BS gleich 0
Hauptspeicher	1100 0000	
c) I_2	0000 0000	BS gleich 0
Hauptspeicher	1111 1111	

7.7. Logische Instruktionen

Die in Abb. 7.9. aufgeführten logischen Instruktionen sind standardmäßig in alle Modelle der Computerfamilie IBM /370 eingebaut.

1) Storage — Immediate Data (Speicher-Direktdaten).
2) BS ist hier die Abkürzung von Bedingungsschlüssel.

Format		RR		RX		SI		SS	
Und	And	14	NR	54	N	94	NI	D4	NC
Vergleichen logisch	Compare Logical	15	CLR	55	CL	95	CLI	D5	CLC
Oder	Or	16	OR	56	O	96	OI	D6	OC
Exklusives Oder	Exclusive Or	17	XR	57	X	97	XI	D7	XC

Abb. 7.9. Logische Instruktionen

Die Instruktionsformate sind bekannt, seien aber hier zur Wiederholung noch einmal aufgeführt (Abb. 7.10.).

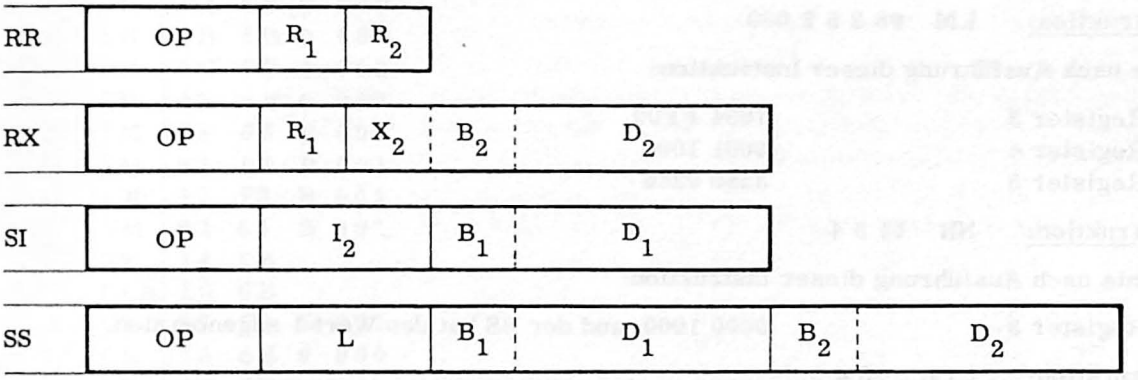


Abb. 7.10. Instruktionsformate bei den logischen Instruktionen

Zu beachten ist hierbei, daß beim SS-Format die beiden Operanden die gleiche Länge besitzen.

Nachstehend folgen einige Bemerkungen zu den einzelnen Instruktionsarten.

1) Und
Der zweite Operand wird durch die logische Und-Operation mit dem ersten Operanden verknüpft. Der BS [1] wird auf 0 gesetzt, wenn das Ergebnis auf allen Bitstellen eine 0 aufweist; anderenfalls wird er auf 1 gesetzt.

2) Oder
Die Verknüpfung der beiden Operanden erfolgt durch die logische Oder-Operation. Der BS wird wie bei der Und-Operation gesetzt.

3) Exklusives Oder
Die beiden Operanden werden durch die logische Operation „Exklusives Oder" verknüpft. Der BS wird genau so gesetzt wie bei der Und-Operation.

4) Vergleichen logisch
Die beiden Operanden werden als Bitmuster betrachtet, und es wird der Reihe nach Bit auf Bit verglichen. Die Vergleichsoperation endet, wenn ungleiche Bits gefunden werden. Der BS wird auf 0 gesetzt, wenn beide Operanden gleich sind, auf 1, wenn der erste Operand kleiner (niedriger) ist als der zweite, und auf 2, wenn der erste Operand größer (höher) ist als der zweite Operand. Diese Instruktion wird vorzugsweise beim Vergleich alphamerischer Daten eingesetzt.

1) BS ist in diesen Abschnitt als Abkürzung von Bedingungsschlüssel gebraucht.

Mit einigen Beispielen wollen wir die Betrachtungen über die logischen Operationen abschließen.

Inhalte vor der Ausführung der Instruktionen:

Speicherplätze 008 600 bis 008 603	7654 FF00
Speicherplätze 008 604 bis 008 607	0001 1000
Speicherplätze 008 608 bis 008 60B	3580 0346
Register 2	0000 8600

1. Instruktion: LM 98 3 5 2 000

Inhalte nach Ausführung dieser Instruktion:

Register 3	7654 FF00
Register 4	0001 1000
Register 5	3580 0346

2. Instruktion: NR 14 3 4

Ergebnis nach Ausführung dieser Instruktion:

Register 3 0000 1000 und der BS hat den Wert 1 angenommen.

3. Instruktion: CLR 15 4 5

Ergebnis nach Ausführung dieser Instruktion: Der BS hat den Wert 1 angenommen.

4. Instruktion: OI 96 06 2 008

Ergebnis nach Ausführung dieser Instruktion:

Speicherplatz 008 608 37 und der BS hat den Wert 1 angenommen.

5. Instruktion: XC D7 01 2 00A 2 000

Ergebnis nach Ausführung dieser Instruktion:

Speicherplätze 008 60A
und 008 60B 7512 und der BS hat den Wert 1 angenommen.

6. Instruktion: XC D7 01 2 00A 2 00A

Ergebnis nach Ausführung dieser Instruktion:

Speicherplätze 008 60A
und 008 60B 0000 und der BS hat den Wert 0 angenommen

Gerade bei den logischen Instruktionen ist es mitunter äußerst nützlich, wenn man den Inhalt der Register und der Speicherplätze als Bitmuster und nicht als Sedezimalziffern niederschreibt, weil man dann um so leichter erkennen kann, zu welchem Ergebnis die betreffende logische Operation führt. Das letzte Beispiel zeigt eine einfache Methode auf, die zum Löschen von Hauptspeicherplätzen (und entsprechend abgewandelt auch zum Löschen von Registern) verwendet werden kann.

Aufgabengruppe 16

1) Gegeben sind:

   a) Inhalt der Speicherplätze 001 800 bis 001 803: 0031 F721
   b) Inhalt des Registers 6: 0000 0017
   c) Inhalt des Registers 12: 0000 1800
   d) BS [1] gleich 3

   Gesucht ist der Inhalt des Registers 6 nach Ausführung der nachstehenden Instruktion! Ferner soll angegeben werden, auf welchen Wert der BS nach Ausführung dieser Instruktionen steht:

a)	LH	48	6B	0	000			
b)	LH	48	6B	0	002			
c)	AH	4A	6B	0	000			
d)	SH	4B	6B	0	002			
e)	MH	4C	6B	0	002			
f)	CH	49	6B	0	002			
g)	LM	98	66	B	000			
h)	TM	91	01	B	001			
i)	TM	91	F3	B	001			
j)	TM	91	64	B	001			
k)	NR	14	6B					
l)	CLR	15	6B					
m)	CLI	95	F0	B	001			
n)	CL	55	6B	0	000			
o)	OR	16	6B					
p)	XR	17	6B					
q)	XC	D7	01	B	002	B	000	
r)	NC	D4	00	B	002	B	003	
s)	OC	D6	00	B	002	B	000	
t)	XC	D7	00	B	002	B	000	

   Bei den Aufgaben ab Aufgabe q) ist der Inhalt der Hauptspeicherplätze 001 800 bis 001 803 gesucht.

2) Unter Benutzung der Daten von Aufgabe 1) ist der Inhalt der Hauptspeicherplätze 001 800 bis 001 803 gesucht, wenn die Instruktion

   a) STM 9Ø 6 6 B ØØØ
   b) MVI 92 6 6 B ØØØ
   c) STH 4Ø 6 Ø B ØØØ

   ausgeführt wurde.

3) Schreibe einen Programmabschnitt, mit dessen Hilfe das Zeichen auf dem Hauptspeicherplatz 001 800 bestimmt werden kann. Das Bitmuster dieses Zeichens soll auf einer Zeile gedruckt werden. Wenn also z. B. auf diesem Platz das Zeichen mit der Sedezimalverschlüsselung C3 steht, so soll die Bitfolge 11000011 gedruckt werden.

---

1) Bei dieser Aufgabengruppe ist für den Bedingungsschlüssel die Abkürzung BS verwendet worden.

4) Modifiziere die Aufgabe 3) so, daß in die Mitte der Bitfolge auch noch das Prüfbit geschrieben wird, und zwar soll ein T geschrieben werden, wenn das Prüfbit 1 ist und ein F, wenn das Prüfbit 0 ist. Beim Beispiel von Aufgabe 3) müßte also 1100T0011 herausgedruckt werden.

5) Schreibe ein Programm, das die Spalten 1 bis 7 einer Karte einliest. Der Inhalt dieser Spalten ist als Oktalzahl aufzufassen (Basis 8). Auf die gleiche Zeile sollen sowohl die eingelesene Oktalzahl als auch die ihr entsprechende Sedezimalzahl gedruckt werden. Wird also z.B. die Oktalzahl 0014762 eingelesen, so sollen die Oktalzahl 0014762 und die ihr entsprechende Sedezimalzahl 006642 ausgedruckt werden.

6) Konvertiere die Oktalzahl von Aufgabe 5) in eine Dezimalzahl und drucke anschließend die sich ergebende Dezimalzahl aus! Beispiel: Aus 0014762_8 ergibt sich 006642_{10}.

7.8. Weitere Verzweigungsinstruktionen

Wir haben bereits die Instruktion „Verzweigen bei Bedingung" kennengelernt, die sowohl im RX-Format als auch im RR-Format auftreten kann (Abschnitt 4.4.). Daneben haben wir die beiden Instruktionen

1)	Verzweigen und Laden Folgeadresse — Branch and Link	45	BAL	RX
2)	Verzweigen u. Laden Folgeadr. (Reg.) — Branch and Link	05	BALR	RR

im gleichen Abschnitt kurz besprochen. Wir erwähnten damals, daß das durch die Instruktionen BAL bzw. BALR angesprochene Register in den letzten 24 Bitstellen die Adresse der nächsten auszuführenden Instruktion, der Folgeinstruktion also, enthält, und daß das linksbündige Byte nicht etwa auf Null gesetzt ist, sondern vielmehr wesentliche Informationen aufweist. Nunmehr können wir diese Aussage konkretisieren. Das angesprochene Register enthält die rechten 32 Bitstellen, d.h. die rechte Hälfte, des laufenden Programmstatuswortes (PSW), so wie es im Basissteuerungs-Modus gebräuchlich ist. Sogar wenn das Programm im erweiterten Steuerungsmodus abläuft, wird die der Basissteuerung entsprechende Form ins Register gebracht. Somit kann der Benutzer zum Instruktionslängenschlüssel, zum Bedingungsschlüssel und zur Programm-Maske zugreifen, falls er es für den Programmablauf für erforderlich hält.

Wenn die Instruktion BAL verwendet wird, so verzweigt das Programm unbedingt zu der Instruktion, die durch die effektive Adresse

$$E_2 = (B_2) + (X_2) + D_2$$

angesprochen wird.

Wird die Instruktion BALR verwendet und ist R_2 ungleich 0, so erfolgt eine Verzweigung zu der Adresse, die auf den 24 Bitstellen niedrigster Ordnung des Registers R_2 steht.

Wird die Instruktion BALR benutzt und ist R_2 gleich Null, so erfolgt keine Verzweigung, das Programm läuft vielmehr mit der nachfolgenden Instruktion weiter. Diese Form dient hauptsächlich dazu, in ein Basisregister einen Wert hineinzustellen. Die Instruktion

BALR R_1, 0

lädt die Adresse des Speicherplatzes ins Register R_1, der eine um 2 höhere Adresse als der Speicherplatz aufweist, auf dem die BALR-Instruktion selbst steht. Anschließend wird die auf die BALR-Instruktion folgende Instruktion ausgeführt. Die erste Instruktion eines Programmes ist gewöhnlich eine solche Instruktion; dadurch können alle anderen Adressen mit Hilfe von Distanzadressen und der ins Register geladenen Basisadresse ausgedrückt und gebildet werden. Auf diese Weise ist es auch möglich, daß ein Programm im Hauptspeicher verschoben werden kann und dennoch richtig arbeitet.

Wir wollen nunmehr einige weitere Verzweigungsinstruktionen kennenlernen. Zunächst wird es sich um zwei Instruktionen handeln, bei denen ein möglicher Sprung an einen Zählvorgang gekoppelt ist.

1)	Verzweigen und Zählen —— Branch and Count	46	BCT	RX
2)	Verzweigen und Zählen (Reg.) —— Branch and Count	06	BCTR	RR

Vom Inhalt des Registers R_1 wird automatisch 1 subtrahiert. Ergibt sich dabei als Resultat Null, so wird das Programm mit der nächsten Instruktion fortgesetzt. Ist jedoch das Resultat ungleich Null, so findet eine Verzweigung statt: bei der Instruktion BCT zum Speicherplatz mit der effektiven Adresse E2 und bei der Instruktion BCTR zu dem Speicherplatz, dessen Adresse im Register R_2 steht. Bei der BCTR-Instruktion erfolgt keine Verzweigung, wenn R_2 gleich Null ist. Somit kann die BCTR-Instruktion dazu verwendet werden, einen im Register R_1 installierten Zähler ständig um 1 zu vermindern, ohne daß jemals eine Verzweigung hervorgerufen wird.

Die beiden Instruktionen „Verzweigen und Zählen" werden sinnvollerweise dann verwendet, wenn es darum geht, eine Instruktionsfolge mehrere Male zu wiederholen. Angenommen, wir wollen ein Programm schreiben, das 20 im Hauptspeicher stehende

Hauptspeicheradressen	Inhalt		
008 700 bis 008 703	X_1	Daten	
008 704 bis 008 707	X_2		
⋮ ⋮	⋮		
008 74C bis 008 74F	X_{20}		
008 800	BALR	05 6 0	Instruktionen
008 802	SR	1B 3 3	
008 804	L	58 2 0 6 016	
008 808	LH	48 5 0 6 01A	
008 80C	A	5A 3 0 2 000	
008 810	SH	4B 2 0 6 01C	
008 814	BCT	46 5 0 6 00A	
008 818 bis 008 81B	00 00 87 4C	(Adresse von X_{20})	
008 81C und 008 81D	00 14	(Zähler)	
008 81E und 008 81F	00 04	(Adreßdifferenz der X-Werte)	

Abb. 7.11. Beispiel einer Instruktionsfolge: Addition von 20 X-Werten

Zahlen X_1, X_2, ···, X_{20} aufeinander addieren und die Summe ins Register 3 stellen soll. Wir können zur Lösung dieses Problems die Instruktionsfolge entwickeln, die in Abb. 7.11. dargestellt ist.

Wir können uns die Programmierung von Schleifen erleichtern, wenn man die beiden nachfolgend beschriebenen Instruktionen verwendet. Beide liegen im RS-Format vor. (Unter einer Schleife versteht man bekanntlich eine Instruktionsfolge, die fortwährend durchlaufen wird.)

3)	Verzweigen bei Index größer —— Branch on Index High	86	BXH	RS
4)	Verzweigen bei Index kleiner oder gleich —— Branch on Index Low or Equal	87	BXLE	RS

Diese beiden Instruktionen arbeiten wie folgt:

a) Der Inhalt des Registers R_3 wird zum Inhalt des Registers R_1 addiert; die Summe ersetzt den Inhalt von R_1.

b) Der neue Inhalt des Registers R_1 wird mit dem Inhalt des Registers R_3 oder mit dem Inhalt des Registers R_3+1 verglichen, je nachdem, welche Registernummer ungerade ist.

c) Die Verzweigung erfolgt bei der BXH-Instruktion, wenn $(R_1) > (R_3)$ ist bzw. wenn $(R_1) > (R_3+1)$ ist und bei der BXLE-Instruktion, wenn $(R_1) \leq (R_3)$ bzw. $(R_1) < (R_3+1)$ ist.

Wenn R_3 ein ungerades Register ist, so wird dieses sowohl im Schritt a) als auch im Schritt b) angesprochen. Gewöhnlich wird man als Register R_3 jedoch ein Register mit gerader Nummer wählen; damit sind aber in diese Instruktionen dann drei Register verwickelt, nämlich R_1, R_3 und R_3+1.

Diese Instruktionen können zur Vereinfachung der Codierung der Schleife des vorigen Beispieles dienen. Man verfolge hierzu die Instruktionsfolge in Abb. 7.12., die die Instruktion BXLE verwendet.

Hauptspeicheradressen	Inhalt	
008 700 bis 008 703	X_1	
008 704 bis 008 707	X_2	
⋮ ⋮	⋮	
008 74C bis 008 74F	X_{20}	
008 800	BALR	05 6 0
008 802	LM	98 2 5 6 00E
008 806	A	5A 3 0 2 000
008 80A	BXLE	87 2 4 6 004
008 810 bis 008 813	00 00 87 00	
008 814 bis 008 817	00 00 00 00	
008 818 bis 008 81B	00 00 00 04	
008 81C bis 008 81F	00 00 87 4C	

Abb. 7.12. BXLE-Instruktion zur Steuerung einer Schleife (siehe hierzu auch Abb. 7.11.)

Zu bemerken wäre noch, wie wir die LM-Instruktion verwendet haben: Wir haben die Adresse von X_1 ins Register 2 geladen, den Anfangswert 0 für die Summe ins Register 3, das Inkrement 4 ins Register 4 und den Vergleichswert, die Endadresse der X-Werte (die Adresse von X_{20}), ins Register 5.

7.9. Stellenverschiebung (Schiften)

Wenn alle Bits eines Registers um eine oder mehrere Stellen nach links oder nach rechts bewegt werden, so bezeichnet man diesen Vorgang als Stellenverschiebung oder Schiften [1]. Führt man eine Verschiebung um eine Stelle durch, so verdoppelt bzw. halbiert man die entsprechende Dualzahl. Die Instruktionen zur Stellenverschiebung besitzen sämtlich das RS-Format. Wie aus der Abb. 7.13. ersichtlich ist, bleibt hierbei ein Halbbyte unberücksichtigt.

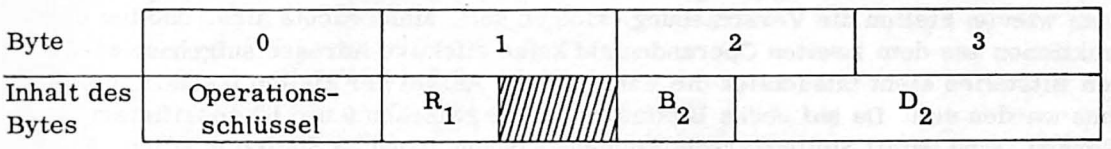


Abb. 7.13. Format der Instruktionen zur Stellenverschiebung

In das vierte Halbbyte dieser Instruktion können beliebige Bits gestellt werden.

Für das Schiften gibt es die in Abb. 7.14. aufgeführten Instruktionen.

Operations-schlüssel	Typ der Versch.	Richtung d. Verschiebung	Umfang der Verschiebung
SRL 88	Logisch	Rechts	Einfach
SLL 89	Logisch	Links	Einfach
SRA 8A	Arithm.	Rechts	Einfach
SLA 8B	Arithm.	Links	Einfach
SRDL 8C	Logisch	Rechts	Doppelt
SLDL 8D	Logisch	Links	Doppelt
SRDA 8E	Arithm.	Rechts	Doppelt
SLDA 8F	Arithm.	Links	Doppelt

Abb. 7.14. Instruktionen zur Stellenverschiebung

Instruktionen, bei denen wir unter dem Verschiebungsumfang „Einfach" notiert haben, befassen sich mit nur einem allgemeinen Register: Es werden die Bits des Register: R_1 verschoben. Instruktionen mit dem Vermerk „Doppelt" müssen sich grundsätzlich auf ein Register R_1 beziehen, das durch eine gerade Nummer gekennzeichnet ist. Es werden die Inhalte von zwei Registern, nämlich von R_1 und von R_1+1, verschoben. Dabei werden die Inhalte beider Register als eine zusammengehörende Information betrachtet, d.h. ,falls diese als Zahl interpretiert wird, so steht das Vorzeichen in $R_1 \cdot R_1+1$ weist dann kein weiteres Vorzeichen auf.

1) Im angloamerikanischen Sprachraum als „shift" bezeichnet.

Wenn unter dem Typ der Verschiebung der Vermerk „Logisch" erscheint, so werden bei diesem Verschiebungstyp alle Bitstellen als gleichwertig angesehen, d.h. alle 32 bzw. 64 Bits werden geschlossen nach rechts oder links verschoben; in die freiwerdenden Bitstellen werden Nullen nachgezogen. Eine arithmetische Stellenverschiebung bewegt nur die 31 bzw. 63 Datenbits, das Vorzeichenbit wird nicht bewegt. Bei den Stellenverschiebungen nach links werden in die freiwerdenden Positionen Nullen nachgezogen; bei den Stellenverschiebungen nach rechts werden jedoch die freiwerdenden Bitstellen mit dem Vorzeichenbit gefüllt (somit kommt es also hier zu einem Nachschieben der Vorzeichen).

Bei den arithmetischen Stellenverschiebungen wird der Bedingungsschlüssel auf 0, 1 oder 2 gesetzt, je nachdem, ob das Ergebnis gleich Null oder negativ oder positiv ist. Der BS[1] wird auf 3 gesetzt, wenn eine Stellenverschiebung nach links bewirkt, daß ein Bit herausgeschoben wird, das verschieden vom Vorzeichenbit ist (Überlaufbedingung). Die logischen Stellenverschiebungen setzen den BS nicht auf einen neuen Wert.

Der Inhalt der sechs rechtsbündig gelegenen Bitstellen des zweiten Operanden (B_2/D_2) gibt an, um wieviel Stellen die Verschiebung erfolgen soll. Man beachte also, daß bei diesen Instruktionen aus dem zweiten Operandenfeld keine effektive Adresse aufgebaut wird. Auf diesen Bitstellen steht tatsächlich die Zahl, die die Anzahl der Stellen festhält, um die verschoben werden soll. Da auf sechs Bitstellen Zahlen zwischen 0 und 63 spezifiziert werden können, sind damit Stellenverschiebungen zwischen 0 und 63 Stellen möglich. Eine arithmetische Stellenverschiebung um 0 Stellen kann dazu benutzt werden, um das Vorzeichen einer Zahl zu bestimmen.

Beispiele:

Inhalte vor Ausführung der Instruktionen:

Register 1	0011 2233
Register 2	FF89 765B
Register 3	AO10 1023

1. Instruktion: SRL 88 1 0 0004

Nach Ausführung dieser Instruktion:

Register 1	0001 1223

2. Instruktion: SRA 8A 2 0 0008

Nach Ausführung dieser Instruktion:

Register 2	FFFF 8976

3. Instruktion: SLDL 8D 2 0 0001

Nach Ausführung dieser Instruktion:

Register 2	FF12 ECB7
Register 3	4020 2046

1) Unter BS wird in diesem Abschnitt der Bedingungsschlüssel verstanden.

7.10. Einige weitere Instruktionen verschiedener Art

In diesem Abschnitt wollen wir uns mit einigen Instruktionen befassen, die jedoch zueinander nicht in Beziehung stehen.

1)

Laden Adresse — Load Address	41	LA	RX

Die effektive Adresse $E_2=(B_2)+(X_2)+D_2$ wird in das Register R_1 hineingestellt. Die acht Bitstellen höchster Ordnung von R_1 werden auf 0 gesetzt. Es sollte ausdrücklich betont werden, daß durch diese Instruktion nicht die auf E_2 gespeicherten Daten ins Register R_1 geladen werden, sondern daß vielmehr die Adresse selbst geladen wird. Diese Instruktion ist außerordentlich wertvoll, wenn es darum geht, kleine Zahlen in ein Register zu laden. So wird z. B. durch

```
LA    41 7 0 0 034
```

ins Register 7 der Wert 0000 0034 gestellt.

2)

Einfügen Zeichen — Insert Character	43	IC	RX

Der Inhalt des durch E_2 adressierten Bytes wird in das rechte Byte des Registers R_1 gestellt. Der Rest des Registers R_1 wird von der Ausführung dieser Instruktion nicht betroffen, er bleibt also unverändert.

3)

Speichern Zeichen — Store Character	42	STC	RX

Das letzte Byte von R_1 wird in das Byte des Hauptspeichers gespeichert, das durch die Adresse E_2 angesprochen wird.

4)

Setzen Programm-Maske — Set Program Mask	04	SPM	RR

Die Bits 2 bis 7 des durch R_1 adressierten Registers ersetzen den Bedingungsschlüssel und die Programm-Maske des laufenden PSW [1]. R_2 wird ignoriert. Der Rest des PSW wird nicht verändert. Im einzelnen ersetzen die Bits 2 und 3 den Bedingungsschlüssel und die Bits 4 bis 7 die Programm-Maske. Die Programm-Maske behandeln wir im Abschnitt 7.13. Vor diese Instruktion wird oft eine BALR- oder BAL-Instruktion codiert, um den Inhalt des laufenden PSW in ein Register zu stellen. Darauf folgen dann logische Und- bzw. Oder-Verknüpfungen, um die entsprechenden (gewünschten) Bits zu verändern.

5)

Supervisor-Aufruf — Supervisor Call	0A	SVC	RR

Für diese Instruktion hat das RR-Format eine besondere Bedeutung (siehe Abb. 7.15.).

Byte	0	1
Inhalt des Bytes	0A	I_1

Abb. 7.15. SVC-Instruktion

1) PSW ist die Abkürzung von „Programmstatuswort".

Von dieser Instruktion werden keine Register berührt. Diese und nur diese Instruktion ruft eine Supervisor-Unterbrechung hervor. Der Inhalt des Bytes I_1 wird in die Bitstellen 24 bis 31 des laufenden PSW gestellt, die Bitstellen 16 bis 23 dieses PSW werden auf 0 gesetzt. Danach wird das laufende PSW als altes PSW auf den Speicherplatz 32 weggespeichert und das neue PSW vom Speicherplatz 96 geladen. Die normale Methode, durch die Problemprogramme mit dem Supervisorprogramm in Verbindung gebracht werden, führt über die SVC-Instruktion. Die in dieser Instruktion stehenden acht Bits von I_1 können gewissermaßen als verschlüsselte Nachrichten angesehen werden; durch diese wird dem Supervisorprogramm gesagt, welche Dienstleistung das Problemprogramm von ihm verlangt.

6)	Addieren logisch Register —— Add Logical	1E	ALR	RR
7)	Addieren logisch —— Add Logical	5E	AL	RX
8)	Subtrahieren logisch Register —— Subtract Logical	1F	SLR	RR
9)	Subtrahieren logisch —— Subtract Logical	5F	SL	RX

Diese vier Instruktionen arbeiten genau so wie die entsprechenden Instruktionen AR, A, SR und S, nur daß sie den BS[1] anders setzen. Sie erleichtern wesentlich die Programmierung der Addition bzw. der Subtraktion von Festpunktdaten, die ein Doppelwort belegen. Der BS wird wie folgt gesetzt:

0	Die Summe ist gleich 0, und auf der Vorzeichenbitstelle kommt es zu keinem Übertrag.
1	Die Summe ist ungleich 0, und auf der Vorzeichenbitstelle kommt es zu keinem Übertrag.
2	Die Summe ist gleich 0, und auf der Vorzeichenbitstelle kommt es zu einem Übertrag.
3	Die Summe ist ungleich 0, und auf der Vorzeichenbitstelle kommt es zu einem Übertrag.

Um die Anwendung dieser Instruktionen kennenzulernen, wollen wir einen typischen Fall betrachten. Angenommen, in den Registern 2 und 3 stehe als Ergebnis einer Multiplikation ein Produkt im Doppelwortformat, ein zweites derartiges Produkt stehe in den Registern 4 und 5. Es soll nun die Summe der beiden Produkte gebildet und in die Register 4 und 5 gespeichert werden. Wir addieren dazu zunächst den Inhalt des Registers 2 auf den Inhalt des Registers 4; anschließend addieren wir logisch den Inhalt des Registers 3 auf den Inhalt des Registers 5. Wird bei der letzten Addition der BS auf 2 oder auf 3 gesetzt, so addieren wir zum Abschluß noch eine 1 zusätzlich auf den Inhalt des Registers 4. Bei diesem Vorgehen haben wir freilich die Möglichkeit eines Überlaufs im Register 4 vernachlässigt. Derartige Berechnungen mit doppelter Genauigkeit sind häufig bei solchen statistischen Berechnungen anzuwenden, bei denen die Summe von Produkten bzw. von Quadraten großer Zahlenmengen gebildet werden muß.

7.11. Privilegierte Instruktionen

Den in diesem Abschnitt zu besprechenden Instruktionen ist gemeinsam, daß sie sämtlich privilegiert sind. Sie können also nur im Supervisorstatus ausgeführt werden. Befindet sich der Computer jedoch im Problemstatus und sollen in diesem

1) BS gilt in diesem Abschnitt als Abkürzung von Bedingungsschlüssel.

Status die zu behandelnden Instruktionen ausgeführt werden, so kommt es zu einer Programmunterbrechung.

1)	Laden Programmstatuswort — Load Program Status Word	82	LPSW	S

Das zweite Byte wird bei dieser Instruktion nicht verwendet. Das durch den ersten Operanden spezifizierte Doppelwort ersetzt das PSW [1]. Als nächste Instruktion wird diejenige ausgeführt, deren Adresse im neuen PSW steht.

2)	Setzen System-Maske — Set System Mask	80	SSM	S

Das zweite Byte dieser Instruktion wird nicht verwendet. Das Byte, auf das die im ersten Operanden festgelegte Adresse Bezug nimmt, ersetzt die Bits der Bitstellen 0 bis 7 des laufenden PSW. Dadurch wird das Supervisorprogramm befähigt, EA-Unterbrechungen zuzulassen oder zu maskieren. Wenn die Bitstelle 1 des Steuerregisters 0 eine 1 enthält, so wird diese Instruktion nicht ausgeführt; sie verursacht vielmehr eine Programmunterbrechung.

3)	Diagnostizieren — Diagnose	83	---	

Für diese Instruktion ist kein mnemonischer Operationsschlüssel festgelegt worden. Die Instruktion führt gewisse diagnostizierende Tests in der ZE[2] aus und zeichnet die Resultate dieser Prüfungen auf, in dem Bytes in Hauptspeicherplätze mit niedrigen Adressen gespeichert werden. Die Ausführung dieser Instruktion erfolgt bei den einzelnen Modellen der Computerfamilie IBM /370 nicht einheitlich; sie ist von Modell zu Modell verschieden. Diese Instruktion ist dazu bestimmt, die Wartung und Instandsetzung des Computers zu unterstützen.

4)	Laden Steuerregister — Load Control	B7	LCTL	RS
5)	Speichern Steuerregister — Store Control	B6	STCTL	RS

Durch die Instruktion LCTL werden die Steuerregister geladen, angefangen beim Register R_1 und aufhörend beim Register R_3. Das Laden erfolgt von den Hauptspeicherplätzen, die bei der effektiven Adresse E_2 beginnen. Umgekehrt werden durch die Instruktion STCTL die Steuerregister R_1 bis R_3 in die bei der effektiven Adresse E_2 beginnenden Hauptspeicherplätze gespeichert. Beide Instruktionen wirken also analog der für die allgemeinen Register geltenden Instruktionen LM und STM (siehe Abschnitt 4.3.), nur daß sie eben die Steuerregister in die Operation einbeziehen.

6)	Schreiben direkt — Write Direct	84	WRD	SI
7)	Lesen direkt — Read Direct	85	RDD	SI

Diese beiden Instruktionen gehören zur speziellen Direktsignal-Steuerung. Sie sind dazu entworfen worden, um eine IBM /370 in die Lage zu versetzen, mit einem anderen Computer Signale auszutauschen. Beide Instruktionen bewirken, daß der zweite Operand I_2 auf acht ausgehenden Signalleitungen in Form von Zeitsignalen in Abständen von $0,5\mu s$ bis $1,0\mu s$ gesendet wird. Die Instruktion WRD stellt das Byte

1) PSW ist die Abkürzung von Programmstatuswort.
2) ZE ist die Abkürzung von Zentraleinheit.

mit der durch den ersten Operanden gebildeten Adresse als statische Signale auf acht anderen Leitungen zur Verfügung; durch Anführung des Begriffes „statisch" meinen wir, daß diese Signale solange bestehen bleiben, bis die nächste WRD-Instruktion gegeben wird. Durch die RDD-Instruktion wird über eine gleiche Gruppe von acht Leitungen ein Byte empfangen und in den Speicherplatz gespeichert, dessen Adresse durch den ersten Operanden festgelegt ist. Normalerweise werden die ausgehenden Signalleitungen dazu verwendet, um externe Unterbrechungen in anderen Computern des Typs IBM /370 hervorzurufen oder um irgendeiner anderen Steuereinheit Signale zu geben.

7.12. Die Hauptspeicherschutzeinrichtung und die für sie geltenden Instruktionen

Die IBM /370 wurde so entworfen, daß sich mehrere Programme zur gleichen Zeit im Hauptspeicher befinden können. Damit ein Programmierirrtum in einem Programm keinen Schaden in den anderen Programmen anrichten kann, gibt es die sogenannte Hauptspeicherschutzeinrichtung.

Jedem aus 2048 Bytes bestehenden Hauptspeicherblock ist ein aus sieben Bits bestehender Speicherschlüssel zugeordnet. Die Bitstellen 0 bis 3 dienen dem Speicherschutz und die Bitstelle 4 dient dem Abrufschutz. Bitstelle 5 enthält ein Referenzbit, Bitstelle 6 ein Änderungsbit. Die Bitstelle 6 wird immer dann auf 1 gesetzt, wenn in irgendein Byte des Blockes eine Einspeicherung erfolgt, sei es durch die Zentraleinheit oder durch einen Kanal. Der Bitstelle 5 wird dagegen stets dann eine 1 zugewiesen, wenn eine Bezugnahme auf irgendein Byte des Blockes erfolgt, sei es durch Schreiben oder durch Lesen.

Jedes Programmstatuswort (PSW) bzw. Kanaladreßwort (CAW) besitzt bekanntlich einen vierstelligen Speicherschutzschlüssel, der die Bitstellen 0 bis 3 einnimmt. Wenn die Bitstelle 4 des Speicherschlüssels eine 0 enthält, so unterliegt jeder Versuch eines Programmes, den Wert eines Bytes im Hauptspeicher zu ändern, allein dem Speicherschutz. Stimmt der Speicherschutzschlüssel im PSW bzw. CAW mit dem Speicherschutz des Speicherschlüssels überein, so erfolgt die Wertänderung, anderenfalls unterbleibt die Wertänderung, und es kommt außerdem zu einer durch den Speicherschutz bedingten Programmunterbrechung. Jedes Programm kann selbstverständlich bei einer 0 auf der Bitstelle 4 jedes Byte des Hauptspeichers auslesen, wobei es gleichgültig ist, ob der Schutzschlüssel im PSW bzw. CAW mit dem im Speicherschlüssel übereinstimmt oder nicht.

Ist das Abrufbit auf Bitstelle 4 des Speicherschlüssels gleich 1, dann ist nicht allein das Speichern, sondern auch das Abrufen (Abholen) von Bytes verboten. Alles in allem bedeutet das ganze also, daß ein Speicherblock entweder gegen unberechtigte Speicherungen und Abrufe geschützt werden kann oder nur gegen unberechtigte Speicherungen.

Von dieser Regel wird abgewichen, wenn der Schutzschlüssel im PSW bzw. CAW aus lauter Nullen besteht. Ein Programm mit einem solchen Nullschlüssel kann zu beliebigen Hauptspeicherstellen zugreifen, sowohl lesend als auch schreibend. Deshalb bleibt der Nullschlüssel normalerweise für Supervisorprogramme reserviert.

Zur Handhabung des Speicherschlüssels dienen die beiden Instruktionen

1)	Setzen Speicherschlüssel —— Set Storage Key	08	SSK	RR

2)	Einfügen Speicherschlüssel —— Insert Storage Key	09	ISK	RR

Das Register R_2 muß bei beiden Instruktionen die Adresse eines aus 2048 Bytes bestehenden Blockes enthalten. Die eigentliche Blockadresse nimmt dabei die Bitstellen 8 bis 20 ein; die Bitstellen 0 bis 7 sowie 21 bis 27 werden ignoriert. Die Bitstellen 28 bis 31 dürfen nur Nullen aufweisen. Das Register R_1 enthält den Speicherschlüssel auf den Bitstellen 24 bis 30, die restlichen Bitstellen des Registers (0 bis 23 bzw. 31) bleiben unbeachtet. Die Instruktion SSK weist den Speicherschlüssel dem Block zu, der durch den Inhalt von R_2 adressiert wird. Die Instruktion ISK hingegen holt den Speicherschlüssel des durch den Inhalt von R_2 spezifizierten Blockes ins Register R_1 hinein.

Im Modus der Basissteuerung werden bei der Instruktion SSK zusätzlich auch die Bitstellen 29 und 30 des Registers R_1 ignoriert. Diese werden immer auf 0 gebracht, desgleichen nehmen auch die Bitstellen 5 und 6 des Speicherschlüssels den Wert 0 an. Die ISK-Instruktion holt bei zugrundegelegter Basissteuerung Nullen ins Register R_1 für die Bitstellen 20 bis 31. Bei der erweiterten Steuerung wird nur die Bitstelle 31 des Registers R_1 durch die Instruktion ISK auf den Wert 0 gebracht.

Beide Instruktionen, SSK wie ISK, sind privilegierte Instruktionen.

Aufgabengruppe 17

1) Gegeben sind folgende Inhalte:

Register 2	0000 0004
Register 3	0000 2024
Register 4	0000 2000
Register 5	FFFF FF13
Register 11	0000 1000
002 000 bis 002 003	0000 8132
002 004 bis 002 007	0000 00F1
002 008 bis 002 00B	0001 E012
002 00C bis 002 00F	0001 EF01
002 010 bis 002 013	FFFF FFD7
002 014 bis 002 017	0000 0103
002 018 bis 002 01B	FFFF F174
002 01C bis 002 01F	0000 0002
002 020 bis 002 023	0000 0008
002 024 bis 002 027	0000 0E00

Gesucht ist der Inhalt der Register 4 und 5 nach Ausführung der nachstehenden Instruktionen:

a) 001 000	SR	1B 55
	A	5A 54 0000
	BXLE	87 42 B002
	ST	50 54 0028
b) 001 000	AR	1A 55
	BCTR	06 2B
	ST	50 53 0004
c) 001 000	L	58 34 0008
	L	58 54 000C
	MR	1C 23
	MR	1C 45
	SLR	1F 53
	BC	47 CB 0014
	BCTR	06 40
	SR	1B 42

d)	SRL	88 40 2000
e)	SLL	89 50 2005
f)	SRA	8A 50 0008
g)	SLA	8B 40 0006
h)	SRDL	8C 40 0007
i)	SLDL	8D 40 B016
j)	SRDA	8E 40 0087
k)	SLDA	8F 40 0020
l)	LA	41 54 B175
m)	IC	43 5B 0012

2) Unter Verwendung der Daten von Aufgabe 1) soll der Inhalt der Speicherplätze 002 024 bis 002 027 nach Ausführung der folgenden Instruktionen angegeben werden:

a)	STC	42 5B B025
b)	TM	93 99 4007
c)	L	58 54 0018
	SRL	88 50 0002
	STC	42 54 0016
	AP	FA 21 4024 4015
d) 001 000	BALR	05 50
	ST	50 54 0024
	OI	96 30 4024
	L	58 54 0024
	SPM	04 50

3) Schreibe eine Supervisoraufruf-Unterroutine, mit der man erreicht, daß nach Ausführung der SVC-Instruktion ein neues PSW vom Speicherplatz 002000 + 8·j geladen wird, wobei j der Unterbrechungsschlüssel ist.

4) Schreibe ein Programm, das einen Kartensatz einliest, dessen einzelne Karten wie folgt abgelocht sind:

Spalten 1 bis 5:	X_1
Spalten 6 bis 10:	X_2
⋮	⋮
Spalten 76 bis 80:	X_{20}

Für jede gelesene Karte soll die Summe der X-Werte und die Summe der Quadrate der X-Werte ausgedruckt werden. Die arithmetischen Operationen sind in der Dualarithmetik auszuführen. Auf den Karten sind grundsätzlich nur Dezimalzahlen abgelocht; sind diese negativ, so enthält das betreffende Feld zusätzlich eine Lochung in Zeile 11 über der Einerstelle.

Für die EA-Operationen sind die mnemonischen Operationsschlüssel einzusetzen, die im Abschnitt 4.5. eingeführt wurden.

5) Schreibe ein Programm, das dieselben Eingabekarten wie bei Aufgabe 4) verwendet. Ausgedruckt werden soll jedoch eine Liste der X- und Y-Werte, wobei die Y-Werte gleich den X-Werten sind, nur daß diese in der dezimalen Gleitpunktform erscheinen sollen. Zur näheren Erläuterung betrachte man die nachfolgenden Beispiele:

X	Y
12345	0.12345E+05
00138	0.13800E+03
00002	0.20000E+01
0000K	-0.20000E+01
00000	0.00000E+00

Für jede gelesene Karte sind zwanzig Ausgabezeilen zu drucken.

6) Aufgabe 5) ist wie folgt abzuändern: Für jede gelesene Karte ist außerdem das mit 10^{-40} multiplizierte Produkt der 20 Zahlen zu bilden und zu drucken. Es kann vorausgesetzt werden, daß das Endergebnis absolut genommen, stets zwischen 10^{-42} und 10^{-38} liegt.

7.13. Programmunterbrechungen

Während der Ausführung der Instruktionen prüft die ZE [1] beständig nach, ob außergewöhnliche Bedingungen auftreten. Bei solchen Bedingungen wie nichtexistierende Hauptspeicheradressen muß es notwendig zu einem Abbruch der Instruktionsausführung kommen. Andere Bedingungen dagegen, wie ein aufgetretener Überlauf, können ignoriert werden oder verursachen eine Unterbrechung; maßgebend hierfür ist allein, welches Vorgehen der Programmierer wünscht.

Das Auftreten von irgendeiner der über 20 verschiedenen möglichen Bedingungen kann eine Programmunterbrechung hervorrufen: Kommt eine Programmunterbrechung zustande, so wird von der ZE ein Unterbrechungsschlüssel gespeichert, aus dem die Art der Unterbrechung ersichtlich ist. Neben diesem Code zeigt der Instruktionslängenschlüssel an, wieviele Halbwörter während der Interpretationsphase der Instruktion (Entschlüsselung der Instruktion) verarbeitet wurden. Die Instruktionsadresse im Programmstatuswort (PSW) vermindert um den doppelten Instruktionslängenschlüssel (zwecks der erforderlichen Angabe der Instruktionslänge in Bytes) ergibt die Adresse derjenigen Instruktion, die die Unterbrechung verursachte. Bei der Basissteuerung wird der Instruktionslängenschlüssel und der Unterbrechungsschlüssel in das PSW auf der Adresse 000 028 (sedezimal) gespeichert. Bei der erweiterten Steuerung ist der Instruktionslängenschlüssel auf den Hauptspeicherplätzen mit den Adressen 000 08C und 000 08D, in einem Halbwort also, zu finden. Von den 16 Bitstellen dieses Halbwortes enthalten die ersten 13 eine Null, die nächsten beiden den eigentlichen Schlüssel in der bekannten Form (siehe Abschnitt 4.11.) und die letzte wieder eine Null. Die Hauptspeicherplätze mit den Adressen 000 08E und 000 08F nehmen den Unterbrechungsschlüssel auf. Man beachte bei der erweiterten Steuerung, daß die Subtraktion des Inhaltes der Halbwörter mit den Adressen 000 02C und 000 08C unmittelbar zur Adresse des Hauptspeicherplatzes führt, auf dem die fehlerhafte Instruktion beginnt.

In einigen Fällen kann die ZE nicht die Instruktionslänge festhalten; es erscheint dann der Instruktionslängenschlüssel 00 (dual). Das passiert bei einigen Modellen, wenn eine lange Datenübertragung erfolgt, z.B. bei Ausführung der Instruktion MVC, und dabei eine Adresse erreicht wird, die entweder nicht existiert oder die zu einem anderen Speicherschlüssel gehört als zu dem des ablaufenden Programmes. Ebenso kann es dazu kommen, wenn ein neues PSW mittels der Instruktion LPSW geladen wird und die durch die Instruktion gebildete Adresse ungültig ist.

1) ZE ist die Abkürzung von Zentraleinheit.

Vier der Bedingungen, die zu einer Programmunterbrechung führen, können maskiert werden, indem die ihnen zugeordneten Bitstellen der Programm-Maske auf 0 gesetzt werden. Ist ein solche Maskierung durchgeführt, so werden die Bedingungen ignoriert, sie gehen verloren.

Nachstehend wollen wir alle Unterbrechungsschlüssel zusammenstellen und ihre Bedeutungen kurz beschreiben. In die Übersicht sind auch diejenigen Schlüssel einbezogen, die für Instruktionen oder Einrichtungen gelten, die wir noch nicht erwähnt haben. Da der Unterbrechungsschlüssel vier Sedezimalziffern einschließt, werden diese gleich mit aufgeführt.

1) Ungültiger Operationsschlüssel ——— 0001
Der in einer Instruktion festgestellte Operationsschlüssel existiert nicht bzw. fehlt bei dem betreffenden Modell der IBM /370. Zum Testen hinsichtlich des Auftretens dieser Bedingung wird im allgemeinen nur das erste Byte jeder Instruktion herangezogen . Der Operationsschlüssel B2 macht hiervon eine Ausnahme; bei seinem Vorhandensein wird auch das 2. Byte untersucht. Wenn ein Programmierer das Auftreten dieses Ausnahmefalls einmal bewußt verursachen will, so sollte er an Stelle des Operationscodes am besten 00 (sedezimal) oder B2E0 (sed) bis B2EF niederschreiben.

2) Privilegierte Operation im Problemstatus ——— 0002
Im Problemstatus wird eine privilegierte Instruktion entdeckt. Die Operation wird unterdrückt.

3) Fehler bei der Instruktion „Ausführen" (EX) ——— 0003
Wenn durch die Instruktion „Ausführen" eine andere Instruktion „Ausführen" angesprochen wird, so wird diese Fehlerbedingung erkannt und angezeigt. (Die Instruktion EX wird im Abschnitt 7.19. beschrieben.)

4) Speicherschutzfehler ——— 0004
Ein Speicherschutzschlüssel wurde verletzt (siehe Abschnitt 7.12.). Die diese Fehlerbedingung hervorrufenden Daten werden weder geholt (abgerufen) noch gespeichert. Der Instruktionslängenschlüssel kann den Wert 0 aufweisen.

5) Adressierungsfehler ——— 0005
Werden Daten oder Instruktionen oder Befehls- oder Kontrollwörter durch Adressen angesprochen, die außerhalb des verfügbaren Hauptspeichers liegen, so wird ein Adressierungsfehler erkannt. Die Operation wird abgebrochen. Ist eine Instruktion von diesem Fehler betroffen, so wird ihre Ausführung unterdrückt. Der Instruktionslängenschlüssel kann gleich 0 sein, wenn die Fehlerbedingung bei Daten auftritt.

Bei bestimmten Modellen der Computerfamilie IBM /370 mit umfangreichen Hauptspeichern kann es jedoch vorkommen, daß ein Hauptspeicherblock im Gegensatz zum übrigen Teil des Hauptspeichers nicht arbeitsfähig ist. In einem solchen Fall gibt es eine Lücke innerhalb des Adreßkreises der verfügbaren Hauptspeicherplätze.

6) Spezifikationsfehler ——— 0006
Ein Spezifikationsfehler wird in folgenden Fällen erkannt:

a) Eine Instruktionsadresse ist nicht durch 2 teilbar, d.h. die Instruktion beginnt nicht auf Halbwortgrenze.

b) Daten, die eine Speicherung in einem Halbwort, Wort oder Doppelwort erfordern, beginnen nicht auf Hauptspeicherplätzen, deren Adresse auf eine Halbwort-, Wort- oder Doppelwortbegrenzung verweist. Im Abschnitt 7.15. werden wir Ausnahmen besprechen, die diese Regel berühren.

c) Bei den Instruktionen ISK und SSK (siehe Abschnitt 7.12.) wird im angesprochenen Register eine Adresse entdeckt, die auf den vier Bitstellen niedrigster Ordnung (rechtsbündige Bitstellen 28 bis 31) keine Nullen aufweist.
d) In einer Instruktion, die sich auf ein Registerpaar mit gerader/ungerader Adresse beziehen muß, wird durch R_1 ein ungerades Register adressiert.
e) Als Adresse eines Gleitpunktregisters wurde nicht 0, 2, 4 oder 6 angegeben.
f) Als Adresse eines für Operationen mit Gleitpunktzahlen erweiterter Genauigkeit vorgesehenen Gleitpunktregisters wurde nicht 0 oder 4 angegeben.
g) Bei der Dezimalarithmetik überschreitet ein Multiplikator oder ein Divisor die maximal zulässige Länge von 15 Ziffernstellen oder aber das Feld des ersten Operanden ist bei einer Multiplikation bzw. Division nicht länger als das zweite Feld.
h) Die Instruktion MC (siehe Kap. 8) weist keine Nullen auf ihren Bitstellen 8 bis 11 auf.
i) Es wird ein PSW geladen, das eine 1 auf der Bitstelle 12 enthält (Kennzeichen für die erweiterte Steuerung), obwohl das betreffende Modell der Computerfamilie IBM /370 nicht mit der erweiterten Steuerung gefahren werden kann.
j) Es wird ein PSW geladen, das die erweiterte Steuerung (Bitstelle 12 enthält eine 1) anzeigt, jedoch auf wenigstens einer der Bitstellen 0, 2 bis 4, 16, 17 und 34 bis 39 keine Null aufweist.

7) Datenfehler ——— 0007
Diese Fehlerbedingung wird erkannt, und die Ausführung der Instruktion wird abgebrochen, wenn folgendes der Fall ist:

a) Bei Dezimaloperationen enthält ein Operand eine ungültige Ziffern- oder Vorzeichenverschlüsselung. Das gleiche gilt auch für Aufbereitungsoperationen [1] und für die durch die Instruktion CVB (Umwandeln ins Duale) bewirkte Operation.
b) Bei Dezimaloperationen überlappen sich Operandenfelder in unzulässiger Weise.
c) Bei der Dezimalarithmetik weist der Multiplikand nicht mindestens L_2 führende Bytes auf, die lauter Nullen enthalten (L_2 ist bekanntlich die Länge des zweiten Operandenfeldes).

8) Überlauf bei der Festpunktarithmetik ——— 0008
Bei einer Addition, einer Subtraktion oder einer Stellenverschiebung mit Festpunktdaten gehen gültige Ziffern verloren bzw. es tritt ein Übertrag entweder auf der Ziffernstelle höchster Ordnung oder auf der Vorzeichenbitstelle auf. Die Operation wird vervollständigt, indem die verlorengegangenen Bits nicht berücksichtigt werden. Die durch diese Fehlerbedingungen hervorgerufene Unterbrechung kann maskiert werden, indem auf die Bitstelle 0 der Programm-Maske (Bitstellen 36 bis 39 im PSW bei der Basissteuerung bzw. Bitstellen 20 bis 23 bei der erweiterten Steuerung) eine 0 gesetzt wird; eine 1 auf dieser Bitstelle läßt die Unterbrechung zu. Läßt man Unterbrechungen zu, so wird jedoch bei einem eintretenden Überlauf der Bedingungsschlüssel nicht auf 3 gesetzt.

9) Festpunkt-Divisionsüberlauf ——— 0009
Diese Fehlerbedingung wird erkannt, wenn bei einer Festpunktdivision die Registergröße überschritten wird, das heißt u.a. auch bei der Division durch 0. Zum Auftreten dieser Bedingung kommt es auch, wenn bei der Instruktion CVB (Umwandeln ins Duale) die entstehende Dualzahl 31 Bitstellen überschreitet. Die Division wird unterdrückt; bei der CVB-Instruktion werden die Bits höchster Ordnung ignoriert, sie gehen einfach verloren.

1) Im angloamerikanischen Sprachraum als „Edit" bezeichnet.

10) Dezimal-Überlauf —— 000A
Diese Fehlerbedingung tritt auf, wenn es während einer Dezimaloperation zu einem Überlauf kommt. Die Operation wird abgeschlossen, indem die Überlaufziffern unberücksichtigt bleiben. Eine Null auf der Bitstelle 1 der Programm-Maske (Bitstellen 36 bis 39 im PSW bei der Basissteuerung bzw. 20 bis 23 bei der erweiterten Steuerung) maskiert diese Unterbrechungsart. Wird die Unterbrechung zugelassen, so wird der Bedingungsschlüssel nicht auf 3 gesetzt.

11) Dezimal-Divisionsüberlauf —— 000B
Diese Fehlerbedingung wird bei der Instruktion DP erkannt, wenn ein Quotient die verfügbare Feldgröße überschreitet. Die Operation wird unterdrückt.

12) Exponentenüberlauf —— 000C
Das Resultat irgendeiner Gleitpunktoperation weist eine Charakteristik auf, die größer als 127 ist. Von einer solchen Charakteristik wird 128 subtrahiert und anschließend die Operation vervollständigt. Das Setzen des Bedingungsschlüssels wird hiervon nicht berührt, da der Exponentenüberlauf das Vorzeichen des Ergebnisses nicht ändert.

13) Exponentenunterlauf —— 000D
Das Resultat irgendeiner Gleitpunktoperation weist eine Charakteristik kleiner als 0 auf (der Exponent ist kleiner als -63). Durch die Bitstelle 2 der Programm-Maske, d.h. Bitstelle 38 des PSW bei der Basissteuerung bzw. Bitstelle 22 bei der erweiterten Steuerung, kann diese Unterbrechungsart maskiert werden (Zuweisen einer 0). Bei einer Maskierung werden sowohl die Charakteristik als auch die Mantisse und die Vorzeichenbitstelle der Gleitpunktzahl (siehe Abb. 7.1.) auf Null gesetzt. Erfolgt die Maskierung nicht, so wird die Operation vervollständigt, indem zuvor 128 auf die Charakteristik addiert wird.

14) Gültigkeitsfehler —— 000E
Diese Fehlerbedingung wird erkannt, wenn bei irgendeiner Gleitpunkt-Operation ein Resultat entsteht, das eine aus lauter Nullen bestehende Mantisse aufweist, während die Charakteristik von Null verschieden ist. Die durch diese Bedingung hervorgerufene Unterbrechung kann maskiert werden, indem die Bitstelle 3 der Programm-Maske (Bitstelle 39 des PSW bei der Basissteuerung bzw. 23 bei der erweiterten Steuerung) auf Null gesetzt wird; in diesem Fall wird der Charakteristik der Wert 0 zugewiesen, wodurch sich eine 0 in Gleitpunktdarstellung ergibt. Wird die Maskenbitstelle auf 1 gebracht, so erfolgt keine Ersetzung der Charakteristik, stattdessen kommt es zum Auftreten der Unterbrechung. Diese Ausnahmebedingung gilt nicht für die Halbierungsinstruktionen HER und HDR.

15) Gleitpunkt-Divisionsüberlauf —— 000F
Diese Fehlerbedingung wird erkannt, wenn der Divisor bei einer Gleitpunktdivision eine aus lauter Nullen bestehende Mantisse aufweist. Die Operation wird unterdrückt.

16) Ausnahme bei der Segmentumsetzung —— 0010
Entweder enthält die Bitstelle für die Segmentverfügbarkeit (Gültigkeitsbitstelle) eine 1 oder die Eintragung in der Segmenttabelle liegt außerhalb des Tabellenumfanges. Im Kapitel 8 werden wir Näheres darüber besprechen.

17) Ausnahme bei der Seitenumsetzung —— 0011
Entweder enthält die Bitstelle für die Seitenverfügbarkeit (Gültigkeitsbitstelle) eine 1 oder die Eintragung in der Seitentabelle liegt außerhalb des Tabellenumfanges. Im Kapitel 8 werden wir Einzelheiten darüber besprechen.

18) Ausnahme bei Spezifikationsumsetzungen ——— 0012
Zu einer Unterbrechung aufgrund einer Ausnahmesituation kommt es in folgenden Fällen:

a) Die Bitstellen 8 und 9 (Seitenumfang) des Steuerregisters 0 enthalten die Bitfolge 00 oder 11
b) Die Bitstellen 8 und 9 des Steuerregisters 0 enthalten die Bitfolge 10, ohne daß der betreffende Computer mit dem Seitenumfang von 4096 Bytes ausgerüstet ist (siehe Abschnitt 3.7.)
c) Die Bitstelle 10 des Steuerregisters 0 enthält eine 1; der Umfang der Seiteneintragungen beträgt jedoch nicht 31 Bitstellen
d) Die Bitstellen 11 und 12 (Segmentumfang) des Steuerregisters 0 enthalten die Bitfolge 01 oder 11
e) Die Bitstellen 11 und 12 des Steuerregisters 0 enthalten die Bitfolge 10; der betreffende Computer ist jedoch nicht mit dem Segmentumfang von 1 048 576 Bytes [1] ausgerüstet.
f) Die Bitstellen 4 bis 7 bzw. 29 bis 30 einer gültigen Eintragung in die Segmenttabelle enthalten keine Nullen.
g) Die eine Bitstelle oder die zwei Bitstellen (abhängig vom Seitenumfang) vor der Bitstelle niedrigster Ordnung einer gültigen Eintragung in die Seitentabelle enthalten keine Nullen (einem Seitenumfang von 2 k Bytes entspricht eine Bitstelle, einem solchen von 4 k Bytes zwei Bitstellen)
h) Die Bitstellen 4 bis 7 einer vier Bytes umfassenden Eintragung in die Seitentabelle enthalten keine Nullen

Die soeben aufgeführten Ausnahmen werden nur erkannt, wenn der Computer mit erweiterter Steuerung gefahren wird oder wenn die Instruktion „Laden der realen Adresse (LRA)" ausgeführt wird; Näheres über diese Instruktion wird im Kap. 8 gesagt.

19) Ausnahme bei einer speziellen Operation ——— 0013
Die Bitstelle 1 des Steuerregisters 1 enthält eine 1 und die Ausführung einer SSM-Instruktion (siehe Abschnitt 7.11.) wurde versucht.

20) Monitor-Ereignis ——— 0040
Die Ausführung der Monitoraufruf-Instruktion (mnemonischer Operationsschlüssel MC) verursacht eine Unterbrechung. Details sind im Kap. 8 zu finden.

21) Programm-Ereignis ——— 0080
Diese Unterbrechungsart wird durch die Maschinenausrüstung „Programm-Ereignis-Registrierung" hervorgerufen. Details sind im Kap. 8 zu finden. Die hierfür bereitstehende Bitstelle des Unterbrechungsschlüssels kann eine 1 zusammen mit anderen Bitstellen des Unterbrechungsschlüssels aufweisen. Sinnvollerweise sorgt man jedoch dafür, daß diese Unterbrechung gewöhnlich allein auftritt. Zu einer Unterbrechung dieser Art kann es nur kommen, wenn der Computer mit der erweiterten Steuerung gefahren wird.

In der Abb. 7.16. sind alle Programmunterbrechungen in einer Tabelle aufgeführt, die außerdem auf die beim Auftreten einer solchen Unterbrechung getroffenen Maßnahmen verweist. Speicherschutzfehler und Adressierungsfehler führen meist zu einem Abbruch während der Instruktionsausführung, wodurch das PSW teilweise oder vollständig verändert vorliegt. Bei gewissen Instruktionen wird jedoch die Instruktionsausführung unterdrückt. Durch Datenfehler bedingte Unterbrechungen führen zu einer

1) Hierfür führt man auch die Abkürzung 1 M Byte (ein Megabyte) ein.

Unterbrechungsschlüssel	Unterbrechung	Instruktionslänge	Ausführung der Instruktion
ØØØ1	Ungültiger Operationsschlüssel	1 2 3	Unterdrückung
ØØØ2	Privilegierte Operation im Problemstatus	1 2	Unterdrückung
ØØØ3	Fehler bei der Instruktion „Ausführung"	2	Unterdrückung
ØØØ4	Speicherschutzfehler	Ø 1 2 3	Unterdrückung oder Abbruch
ØØØ5	Adressierungsfehler	Ø 1 2 3	Unterdrückung oder Abbruch
ØØØ6	Spezifikationsfehler	Ø 1 2 3	Unterdrückung oder Vervollständigung
ØØØ7	Datenfehler	2 3	Unterdrückung oder Abbruch
ØØØ8	Überlauf bei der Festpunktarithmetik	1 2	Vervollständigung
ØØØ9	Festpunkt-Divisionsüberlauf	1 2	Unterdrückung oder Vervollständigung
ØØØA	Dezimalüberlauf	2 3	Vervollständigung
ØØØB	Dezimal-Divisionsüberlauf	2 3	Unterdrückung
ØØØC	Exponentenüberlauf	1 2	Vervollständigung
ØØØD	Exponentenunterlauf	1 2	Vervollständigung
ØØØE	Gültigkeitsfehler	1 2	Vervollständigung
ØØØF	Gleitpunkt-Divisionsüberlauf	1 2	Unterdrückung
ØØ1Ø	Ausnahme bei der Segmentumsetzung	1 2 3	Nichtaufsetzung
ØØ11	Ausnahme bei der Seitenumsetzung	1 2 3	Nichtaufsetzung
ØØ12	Ausnahme bei Spezifikationsumsetzungen	1 2 3	Unterdrückung
ØØ13	Ausnahme bei einer speziellen Operation	2	Unterdrückung
ØØ4Ø	Monitor-Ereignis	2	Vervollständigung
ØØ8Ø	Programm-Ereignis	Ø 1 2 3	Vervollständigung

Anmerkungen: 1) *Nichtaufsetzung* bedeutet, daß der Instruktionslängenschlüssel nicht geändert wird. Die Operation unterbleibt selbstverständlich.
2) *Unterdrückung* heißt, daß die Operation nicht ausgeführt wurde. Der Instruktionslängenschlüssel wurde jedoch geändert.
3) *Abbruch* bedeutet, daß die Instruktion teilweise ausgeführt wurde.

Abb. 7.16. Die Schlüssel der Programmunterbrechungen und die Folgemaßnahmen

Unterdrückung der Operation, wenn das Vorzeichen ungültig ist; anderenfalls werden sie abgebrochen. Bei durch Programmereignisse bedingten Unterbrechungen wird die Operation vervollständigt; es sei denn, daß Unterbrechungen anderer Art gleichzeitig auftreten. In solchen Fällen herrschen die anderen Unterbrechungsarten vor. Eine durch einen Gültigkeitsfehler hervorgerufene Unterbrechung unterdrückt die Ausführung der Instruktion; eine Operation, die ein neues PSW ladet, wird jedoch vervollständigt.

Der Grund dafür, daß vier Bedingungen maskiert werden können, ist darin zu suchen, daß diese in manchen Fällen erwartet werden und daß der Programmierer deshalb in der Lage sein muß, sie ohne besondere Vorkehrungen zu behandeln. Im allgemeinen wird man jedoch allen Bitstellen der Programm-Maske eine 1 zuweisen, wodurch alle Unterbrechungen zugelassen sind. Für die Auswertung der Unterbrechungen wird man dann eine Fehlerbehandlungsroutine schreiben, die die Bearbeitung aller Fehlerbedingungen ermöglicht.

7.14. Andere Unterbrechungsarten

Wir haben bereits die EA-Unterbrechungen und die Programmunterbrechungen besprochen. Mit den übrigen Unterbrechungsarten wollen wir uns jetzt beschäftigen.

Bleiben wir zunächst bei der durch den Supervisoraufruf (SVC-Instruktion) bedingten Unterbrechung. Bei einer solchen wird einfach der Inhalt des zweiten Bytes der Instruktion in das zweite Byte des Unterbrechungsschlüssels hineingestellt. Bei der Basissteuerung wird dann das Programmstatuswort auf das bei 000020 (sed) beginnende Doppelwort gestellt und das neue PSW vom Doppelwort mit der Adresse 000 060 geladen. Im alten PSW steht der Unterbrechungsschlüssel bekanntlich auf den Bitstellen 16 bis 31. Da nur die Bitstellen 24 bis 31 zur Aufnahme des zweiten Bytes der Instruktion benötigt werden, werden die Bitstellen 16 bis 23 zusätzlich auf Null gesetzt. Im erweiterten Steuerungsmodus erfolgt zusammen mit dem Wegspeichern des alten PSW die Speicherung verschiedener relevanter Informationen auf die festen Hauptspeicherplätze mit den Adressen 000 088 bis 000 08B, d.h. auf ein Wort. Ihre Bitstellen 0 bis 12 sowie 15 bekommen Nullen zugewiesen, die Bitstellen 13 und 14 enthalten den

Unterbrechungs-schlüssel (sed)	Ursache
1005	Zeitgeber in der Zentraleinheit
1004	Uhrzeitvergleicher
0080	Intervallzeitgeber
0040	Unterbrechungstaste
0020	Externes Signal 2
0010	Externes Signal 3
0008	Externes Signal 4
0004	Externes Signal 5
0002	Externes Signal 6

Abb. 7.17. Ursachen externer Unterbrechungen

Instruktionslängenschlüssel, die Bitstellen 16 bis 31 den Unterbrechungsschlüssel, wobei die ersten acht Bitstellen wiederum nur Nullen aufweisen.

Der Restartunterbrechung werden wir im Detail im nächsten Kapitel begegnen. Durch sie wird das alte PSW auf das Doppelwort mit der Adresse 000 008 gebracht und das neue PSW vom Doppelwort mit der Adresse 000 000 geholt (beide Adreßangaben sind in sedezimaler Form gemacht).

Bei einer externen Unterbrechung im erweiterten Steuerungsmodus werden wesentliche Informationen ins Wort mit der sedezimalen Adresse 000 084 gespeichert, wobei die ersten beiden Speicherplätze gelöscht werden. Auf das zweite Halbwort mit der Adresse 000 086 wird der Unterbrechungsschlüssel gebracht. Bei der Basissteuerung wird der Unterbrechungsschlüssel ins alte PSW gespeichert (Bitstellen 16 bis 31). Der Inhalt der Hauptspeicherplätze 000 084 bis 000 087 wird nicht zerstört. Den verschiedenen Arten externer Unterbrechungen sind die in der Abb. 7.17. aufgeführten Unterbrechungsschlüssel zugeordnet.

Unterbrechungs-schlüssel (sed)	Ursache
1005	Zeitgeber in der Zentraleinheit
1004	Uhrzeitvergleicher
0080	Intervallzeitgeber
0040	Unterbrechungstaste
0020	Externes Signal 2
0010	Externes Signal 3
0008	Externes Signal 4
0004	Externes Signal 5
0002	Externes Signal 6
0001	Externes Signal 7

Abb. 7.17. Ursachen externer Unterbrechungen

Basiert eine externe Unterbrechung auf mehreren auslösenden Momenten, so werden die einzelnen Bits durch die logische Oder-Operation miteinander verknüpft.

Die Bitstelle 7 des PSW ist für die Maskierung aller externen Unterbrechungen vorgesehen. Enthält diese Bitstelle eine 0, so sind alle externen Unterbrechungen gesperrt. Sie bleiben genau so hängen wie die EA-Unterbrechungen. Enthält dagegen diese Bitstelle eine 1, so schlagen bei der Basissteuerung alle externen Unterbrechungen durch. Bei der erweiterten Steuerung werden bei einer 1 auf der Bitstelle 7 des PSW einige Bitstellen im Steuerregister 0 zusätzlich untersucht. Von ihrem Inhalt hängt es dann ab, ob die betreffenden Unterbrechungsarten zugelassen sind oder nicht. Folgende Bitstellen des Steuerregisters 0 spielen hierbei eine Rolle:

Bitstelle	Maske für
20	Uhrzeitvergleicher
21	Zeitgeber in der Zentraleinheit
24	Intervallzeitgeber
25	Unterbrechungstaste
26	Alle externen Signale

Die erweiterte Steuerung bietet uns also eine zweistufige Maskierung für einige Unterbrechungsarten an. Die hier nicht aufgeführten Arten werden einstufig in der gleichen Form behandelt wie bei der Basissteuerung.

Mit dem Uhrzeitvergleicher, dem Zeitgeber in der Zentraleinheit und dem Intervallzeitgeber wollen wir uns im nächsten Kapitel auseinandersetzen.

Das Drücken der Unterbrechungstaste am Konsol verursacht die gleichbenannte Unterbrechung. Die externen Signale 2 bis 7 sind für Geräte außerhalb des Computers vorgesehen, damit sie den Computer ansprechen können. Bei den meisten Modellen der Computerfamilie IBM /370 ist der Anschluß externer Signalleitungen an die Zentraleinheit nur mittels einer gesonderten Ausrüstung möglich. Bei einigen größeren Modellen sind derartige Vorkehrungen bereits von Haus aus getroffen.

7.15 Byteorientierte Operanden

Wir haben bereits eine Reihe von Instruktionen beschrieben, für die wir statuiert haben, daß bei ihnen gewisse in die Operation eingehende Operanden auf Halbwort-, Wort- oder Doppelwortbegrenzung beginnen sollten. Unter bestimmten Umständen können wir diese Restriktion vergessen.

Bei allen nichtprivilegierten Instruktionen mit Ausnahme der beiden Instruktionen CS und CDS, zwei Instruktionen, denen wir noch nicht begegnet sind, können wir tatsächlich die Begrenzungsvorschriften für die Operanden (Halbwort-, Wort-, Doppelwortbegrenzung) außer acht lassen. Die entsprechenden Operanden dürfen auf jeder beliebigen Speicherstelle beginnen. Wenn jedoch ein solcher Operand nicht so im Hauptspeicher angeordnet ist, wie es seiner Natur entspricht, d.h. wenn er nicht an der ihm zugeordneten normalen Speicherbegrenzung anfängt, so wächst u.U. die Ausführungszeit für die Instruktion beträchtlich an. Bleiben wir zur Illustration unserer soeben gegebenen Behauptung bei einem Beispiel, das aus der Instruktionsfolge

— Laden Wort (L)
— Addieren eines Wortes (A)
— Speichern Wort (ST)

bestehen soll. Die totale Operationszeit liegt bei den einzelnen Modellen um folgende Prozentsätze höher:

- Modell 135: 144%
- Modell 145: 91%
- Modell 165: 1287%

Vorausgesetzt ist bei dieser Berechnung ferner, daß die in die drei Operationen eingehenden Speicheradressen jeweils ein Byte nach der Wortgrenze ansprechen. Es sei nochmals betont, daß Programmunterbrechungen, die auf Spezifikationsfehler zurückzuführen wären, hierbei niemals auftreten können.

Die Begrenzungsregeln bezüglich der Anordnung von Operanden müssen allerdings bei den privilegierten Instruktionen beachtet werden, ferner auch bei den beiden nichtprivilegierten Instruktionen CS und CDS. Wir sollten an dieser Stelle auch noch daran erinnern, daß alle Instruktionen nur an Halbwortbegrenzungen beginnen dürfen. Ebenso müssen Kanalbefehlswörter den Doppelwörtern im Hauptspeicher angepaßt sein.

7.16. Instruktionen und Masken

Drei Instruktionen gibt es, für die das RS-Format in einer speziellen Form voliegt (Abb. 7.18.).

Byte	0	1		2		3
Inhalt	Operationsschlüssel	R_1	M_3	B_2	D_2	

Abb. 7.18. RS-Format der maskenbezogenen Instruktionen

Normalerweise bezieht sich der dritte Operand bei den Instruktionen im RS-Format auf ein Register; allgemein bisher mit R_3 bezeichnet. Bei den jetzt zu besprechenden drei Instruktionen dient der dritte Operand als Maske , deswegen auch M_3 benannt. Jede Bitstelle des Bytes M_3 legt fest, wie das entsprechende Byte des Registers R_1 in die Operation einbezogen wird (4 Bitstellen $\triangleq$ 4 Bytes). Die effektive Adresse $E_2 = (B_2) + D_2$ führt auf einen Hauptspeicherplatz, von dem aus soviele Bytes in die Operation eingehen wie Einsen in der Maske enthalten sind. Die drei Instruktionen heißen:

a)	Einfügen von Zeichen unter einer Maske — Insert Characters Under Mask	BF	ICM
b)	Speichern von Zeichen unter einer Maske — Store Characters Under Mask	BE	STCM
c)	Vergleichen von Zeichen unter einer Maske — Compare Logical Characters Under Mask	ED	CLM

Durch die Instruktion ICM werden diejenigen Bytes im Register R_1 mit neuen Inhalten versorgt, denen ein 1-Bit im M_3 entspricht. Der Ursprung der neuen Inhalte sind dabei die Hauptspeicherstellen, die bei E_2 beginnen. Der Bedingungsschlüssel wird auf 0 gesetzt, wenn entweder die Maske M_3 gleich 0000 (dual) ist oder wenn alle eingefügten Zeichen nur 0-Bits enthalten. Dem Bedingungsschlüssel wird der Wert 1 zugewiesen, wenn die erste Bitstelle des ersten eingefügten Zeichens eine 1 enthält. In allen anderen Fällen wird der Bedingungsschlüssel auf 2 gesetzt. Wenn das Maskenhalbbyte M_3 aus lauter Einsen besteht, so werden alle Bytes des Registers R_1 neu formiert; der Bedingungsschlüssel zeigt dann auf, daß der neue Registerinhalt gleich Null, negativ oder positiv ist.

Durch die Instruktion STCM werden diejenigen Zeichen von R_1, denen ein 1-Bit von M_3 entspricht, auf Hauptspeicherstellen weggespeichert, die bei der Adresse E_2 beginnen. Der Bedingungsschlüssel bleibt ungeändert.

Die Instruktion CLM arbeitet prinzipiell genau so wie die Instruktionen CL oder CLC. Der erste Operand besteht jedoch aus denjenigen Bytes des Registers R_1, denen 1-Bits in M_3 entsprechen. Der Bedingungsschlüssel wird auf 0 gesetzt, wenn entweder die Maske M_3 aus lauter Nullen besteht oder wenn beide Operanden gleich sind. Ihm wird der Wert 1 zugewiesen, wenn der erste Operand kleiner als der zweite ist; der Wert 2 zeigt an, daß der zweite Operand der kleinere von beiden ist.

Mit einigen Beispielen wollen wir diesen Abschnitt beschließen:

Inhalte vor der Ausführung der Instruktionen:

Register 3	00 03 30 00
Register 4	09 08 87 06
Speicherplätze 033 000	FF FE 93 27 46 87
bis 033 005	

1. Instruktion: CLM BD 42 3 005

Durch diese Instruktion wird der Bedingungsschlüssel auf 0 gesetzt, da in den Vergleich das dritte Byte des Registers 4 und das Byte mit der Adresse 033005 einbezogen werden. Beide Bytes weisen aber den gleichen Inhalt, nämlich 87 (sed) auf.

2. Instruktion: STCM BE 43 3 004

Inhalt nach Ausführung dieser Instruktion:

Speicherplätze 033 000 bis 033 005 | FF FE 93 27 87 06

3. Instruktion: ICM BF 4F 3 000

Inhalt nach Ausführung dieser Instruktion:

Register 4 | FF FE 93 27

Außerdem wird dem Bedingungsschlüssel eine 1 zugewiesen.

7.17. Übersetzungsinstruktionen

Zur Übersetzung von Informationen von einem 8-Bit-Code in einen anderen können wir die nachstehend beschriebene Instruktion einsetzen.

1)

Übersetzen — Translate	DC	TR	SS

Die durch den zweiten Operanden angegebene Adresse bezieht sich auf den niedrigsten Speicherplatz einer aus 256 Bytes bestehenden Tabelle. Das erste Operandenfeld hat eine Länge zwischen 1 und 256 Bytes. Der Inhalt jeden Bytes des ersten Operandenfeldes wird als positive ganze Dualzahl betrachtet, die zur effektiven Adresse $E_2 = (B_2) + D_2$ des Tabellenanfanges addiert wird; auf diese Weise wird ein ganz bestimmtes Byte der Tabelle angesprochen. Der Inhalt des angesprochenen Bytes ersetzt anschließend den Inhalt des zur Adressbildung herangezogenen Bytes des ersten Operandenfeldes. Die Operation beginnt mit dem ersten Byte des ersten Operandenfeldes und fährt Byte auf Byte solange fort, bis das erste Operandenfeld erschöpft ist, d.h. der Längenschlüssel L abgearbeitet ist.

Wir sehen also, daß die TR-Instruktion die Umwandlung von Informationen eines 8-Bit-Codes in einen anderen erlaubt.

Eine ähnliche Instruktion ist die folgende:

2)

Übersetzen und Prüfen — Translate and Test	DD	TRT	SS

Durch den zweiten Operanden wird wie bei der Instruktion TR eine 256 Bytes umfassende Tabelle spezifiziert. Das Prüfen schreitet seriell vorwärts und beginnt mit dem ersten Byte des ersten Operandenfeldes, also mit dem Byte, das im ersten Operandenfeld die niedrigste Adresse besitzt. Die Tabellenbytes werden genau so angesprochen wie bei der Instruktion TR. Zum Unterschied gegenüber der Instruktion TR werden jedoch die Bytes des ersten Operandenfeldes nicht verändert, d.h. durch Bytes aus der Tabelle ersetzt. Wenn das durch das Byte des ersten Operandenfeldes adressierte Tabellenbyte 00 enthält, so wird die Operation mit dem nächstfolgenden Byte des ersten Operandenfeldes fortgesetzt. Das erste gefundene Byte, das nicht auf 00 steht, bewirkt folgende zwei Teiloperationen:

a) Die Adresse des Bytes des ersten Operanden, das auf das von 0 verschiedene Tabellenbyte gestoßen ist, wird in das allgemeine Register 1 gestellt, und zwar rechtsbündig auf die Bitstellen 8 bis 31.

b) Der Inhalt des gefundenen von Null verschiedene Tabellenbytes wird auf die Bitstellen 24 bis 31 des allgemeinen Registers 2 gestellt.

BS [1]	Bedeutung
0	Die Operation ist dadurch abgeschlossen worden, daß alle Bytes des ersten Operandenfeldes als Suchargument dienten, jedoch keines von ihnen auf ein Tabellenbyte ungleich 0 gestoßen ist.
1	Zur Prüfung wurden nicht alle Bytes des ersten Operandenfeldes herangezogen, denn irgendein Byte ist auf ein von Null verschiedenes Tabellenbyte gestoßen.
2	Das dem letzten Byte des ersten Operandenfeldes entsprechende Tabellenbyte enthält einen von 0 verschiedenen Wert.

Oberflächlich gesehen, ähnelt die TRT-Instruktion der TR-Instruktion. Von der Anwendungsseite her betrachtet, existiert jedoch ein erheblicher Unterschied: Die TRT-Instruktion wird dazu verwendet, um Eingabedaten mit dem Ziel abzutasten, die Stellung eines bestimmten Zeichens, beispielsweise eines Dezimalpunktes, zu bestimmen. Somit können Informationen, die in völlig freier Form in den Computer eingelesen werden, leicht in Sätze, Gruppen usw. zerlegt werden. Um das dabei zweckmäßigerweise anzuwendende Vorgehen zu illustrieren, wollen wir ein Beispiel verfolgen.

Nehmen wir an, daß in den Hauptspeicher eine elfstellige Zahl mit einem Dezimalpunkt eingelesen wurde. Einige derartige Zahlen sollen jetzt zur Demonstration aufgeführt werden:

```
12345678.01
12.34567908
.1110003333
0000000011.
```

Wir wollen nun das auf den Speicherplätzen 008 E14 bis 008 E1E gespeicherte aus 11 Bytes bestehende Feld in ein Feld mit 20 Bytes übertragen. Die Bytes dieses Feldes sollen die Adressen 008 E2A bis 008 E3D besitzen. Der Dezimalpunkt soll nicht mit übertragen werden; außerdem sollen die rechten 10 Bytes des Zielfeldes die nach dem Komma stehenden Ziffern (Dezimalstellen) der Dezimalzahl enthalten. Das heißt z.B., daß 12345678.01 übergehen soll in

```
00123456780100000000
```

Verfolgen wir also die Speicherbelegung und die Instruktionen in der Abb. 7.19.

Wollen wir das Ergebnis in gepackter Form erhalten, so können wir anschließend eine PACK-Instruktion geben. Zum Beispiel ist noch zu bemerken, daß wir einige Anstrengungen unternehmen mußten, um die richtige Adresse für den ersten Operanden der letzten Instruktion zu bestimmen. Wir eruierten sie mittels arithmetischer Operationen im Register 2. Ein solches Vorgehen pflegt man als Adreßrechnung oder Adreßmodifikation zu bezeichnen. Die auf den Speicherplätzen 008 E1F bis 008 E29 stehenden Nullen haben für die Lösung unseres Problems keine Bedeutung; sie sind nur aufgeführt, um eine optisch deutlichere Trennung der beiden Felder (X bzw. Y) zu erreichen.

7.18. Aufbereitung

Sicherlich ist es meist sinnvoller, eine benannte Zahl, wie z.B. $12,345.67 entweder in die duale Form oder in die gepackte Dezimalform zu überführen, wenn man mit

1) BS ist die Abkürzung von Bedingungsschlüssel

Hauptspeicheradressen	Daten
008 E00 bis 008 E13	Zwanzig Bytes, jedes enthält 00
008 E14 bis 008 E1E	X-Wert (Ausgangsfeld)
008 E1F bis 008 E29	Elf Bytes, jedes enthält 00
008 E2A bis 008 E3D	Y-Wert (Zielfeld)
008 F00 bis 008 FFF	Tabelle; alle Bytes enthalten 00 mit Ausnahme des Bytes mit der Adresse 008 F4B. (Der Punkt ist im EBCDI-Code mit 4B verschlüsselt).

Register	Inhalt
1	0000 0000
2	0000 0000
3	0000 8E00

Instruktionen

MVC	D2 13 3 02A 3 000	Y wird auf 0 gesetzt
TRT	DD 0A 3 014 3 100	Die Adresse des Punktes wird ins Register 1 gestellt
MVC	D2 09 1 000 1 001	Die Dezimalstellen von X werden um eine Stelle nach links verrückt
LR	18 2 3	} Adreß-Modifikation
SR	1B 2 1	
AR	1A 2 3	
MVC	D2 09 2 048 3 014	

Abb. 7.19. Beispiel für den wirkungsvollen Einsatz der Instruktion TRT

ihnen rechnen will oder muß. Dabei wird man sich die Stellung des Dezimalpunktes merken müssen, jedoch wird man den Dezimalpunkt selbst nicht speichern, d.h. in gepackter Dezimalform wird die obige Zahl zu 12 34 56 7+ gespeichert. Muß man jedoch derartige Werte so drucken lassen, daß sie bequem lesbar sind, so bleibt nichts anderes übrig, als Punkt, Komma und Währungszeichen (beim obigen Beispiel) in das zu druckende Feld einzufügen. Ein derartiges Vorgehen bezeichnet man als Aufbereitung.

Für die Aufbereitung sind zwei Instruktionen vorgesehen, die die Umwandlung von gepackten Dezimalfeldern in Felder mit ungepackter Zeichendarstellung unter Einfügung zusätzlicher Zeichen erleichtern.

1)	Aufbereiten — Edit	DE	ED	SS

2)	Aufbereiten und markieren — Edit und Mark	DF	EDMK	SS

Bei beiden Instruktionen wird vorausgesetzt, daß der zweite Operand sich auf ein gepacktes Dezimalfeld bezieht. Die Längenangabe im zweiten Byte der Instruktion gilt für den ersten Operanden; dieser spezifiziert ein Feld, daß vor der Durchführung der

Aufbereitung eine sogenannte Schablone enthalten muß. Das Schablonenfeld ist ungepackt und kann alle darstellbaren Zeichen enthalten, z. B. also Punkte, Kommas usw.,

Sed. Verschlüsselung	Dtsch. Bezeichn.	Engl. Bezeichn.	Abkürzung	Graphisches Symbol
20	Ziffernauswahlzeichen	Digit Select	DS	▽
21	Startzeichen für geltende Ziffern	Significance Start	SOS	[
22	Feldtrennungszeichen	Field Separator	FS	]

Abb. 7.20. Aufbereitungszeichen

die in das Resultatfeld eingefügt werden sollen [1]. Drei spezielle Aufbereitungszeichen werden im Zusammenhang mit diesen Instruktionen gebraucht. Sie haben eine besondere Bedeutung und erfüllen eine bestimmte Funktion (siehe hierzu Abb. 7.20.).

Das erste Zeichen eines Schablonenfeldes wird außerdem als Füllzeichen bezeichnet.

Angenommen, im Hauptspeicher stehe ein gepacktes Dezimalfeld, dessen Inhalt 00 12 34 56 7+ ist. Dieses Feld soll so aufbereitet werden, daß wir seinen Inhalt nach der Ausgabe zu bbb12 345.67bbb lesen können [2]. Um dies zu erreichen, beginnen wir mit der Zusammenstellung der Schablone. In diesem Fall muß sie folgenden Aufbau besitzen:

b▽,▽▽▽,▽▽[,[[bCR

Bei dieser Schablone ist b als Füllzeichen verwendet worden. Hier steht b wie gewöhnlich für das Leerzeichen, CR ist ein Kreditsymbol, das rechts im Anschluß an die Ziffern nach einem Leerzeichen erscheinen soll, wenn die aufzubereitende Zahl negativ ist. Der Computer führt die ED-Instruktion in folgenden Einzelschritten aus (zugrundegelegt ist dieser Beschreibung unser Beispiel):

a) Das Füllzeichen links bleibt unberührt; es wird jedoch außerdem auch für den zukünftigen Gebrauch aufbewahrt.

b) Immer dann, wenn ein ▽-Zeichen in der Schablone erscheint, wird eine Ziffer vom zweiten Operandenfeld abgeholt. Ist diese Ziffer eine führende Null, so ersetzt das Füllzeichen das Schablonenzeichen, anderenfalls tritt die abgeholte Ziffer an Stelle des Schablonenzeichens. Bei unserem Beispiel ist die erste Ziffer im zweiten Operandenfeld eine 0, infolgedessen wird das ▽-Zeichen durch das Füllzeichen ersetzt.

1) Die in der Schablone stehenden Zeichen heißen Aufbereitungszeichen.
2) b bedeutet in diesem Abschnitt das Leerzeichen.

c) Der Computer geht zum nächsten Byte über. In diesem Fall stößt er in der Schablone auf das Komma. Da dieses Zeichen weder ein Ziffernauswahlzeichen noch ein Startzeichen für geltende Ziffern ist, stellt der Computer fest, ob er bereits eine gültige Ziffer gefunden hatte. Da bisher nur führende Nullen im aufzubereitenden Feld entdeckt wurden, so wird an Stelle des Kommas das Füllzeichen in das erste Operandenfeld eingefügt.

d) Das nächste Aufbereitungszeichen ist ein ▽ ; diesem ▽ entspricht im zweiten Operandenfeld eine 0. Deshalb wird auch an dieser Stelle das Füllzeichen und nicht die 0 in das erste Operandenfeld gesetzt.

e) Das nächste Zeichen in der Schablone ist wiederum ein ▽ , dem aber dieses Mal die erste gültige Ziffer, nämlich die 1, des zweiten Operandenfeldes entspricht. Da nunmehr die erste gültige Ziffer entdeckt wurde, stellt der Computer die 1 an die Stelle des Ziffernauswahlzeichens und merkt sich außerdem, daß die erste gültige Ziffer aufgetreten ist.

f) Nachfolgende Aufbereitungszeichen, die Ziffernauswahlzeichen oder Startzeichen für geltende Ziffern sind, werden grundsätzlich durch die entsprechenden Ziffern des zweiten Operandenfeldes ersetzt. Alle in der Schablone stehenden anderen Zeichen, die also nicht gleich den ▽- und [-Zeichen sind, bleiben ungeändert erhalten. Dieser Ablauf setzt sich solange fort, bis im zweiten Operandenfeld das Vorzeichen gefunden wird.

g) Enthält das zweite Operandenfeld einen positiven Wert, wie es auch bei unserem Beispiel der Fall ist, so werden die restlichen Aufbereitungszeichen des ersten Operandenfeldes durch das Füllzeichen ersetzt. Ist der Inhalt des aufzubereitenden Feldes jedoch negativ, so werden die restlichen Zeichen des Schablonenfeldes nicht geändert, sondern bleiben erhalten.

Der Bedingungsschlüssel wird auf 0 gesetzt, wenn das zweite Operandenfeld eine gepackte Dezimalzahl enthält, die gleich Null ist; er wird auf 1 gesetzt, wenn der Inhalt des zweiten Operandenfeldes kleiner als 0 ist, und auf 2, wenn er größer als Null ist.

Im allgemeinen wird nur ein einziges Feld mit einem Male aufbereitet. Es ist jedoch möglich (wenn auch nicht immer einfacher), mit einer einzigen Instruktion mehrere Felder aufzubereiten. Die aufzubereitenden Felder müssen dabei aneinander stoßen, und das Feldtrennungszeichen muß die Schablonen für die einzelnen Felder voneinander trennen. Das Feldtrennungszeichen wird stets als erstes Zeichen einer Schablone betrachtet und demzufolge grundsätzlich immer durch das allererste Zeichen der Schablone, durch das Füllzeichen also, ersetzt. Wird eine Schablone benutzt, die als erstes Zeichen ein Ziffernauswahlzeichen oder ein Startzeichen für geltende Ziffern enthält, so wird dieses Zeichen immer durch die erste Ziffer des zweiten Operandenfeldes ersetzt, selbst wenn diese gleich Null ist.

Das Startzeichen für geltende Ziffern bewirkt zum Unterschied gegenüber dem Ziffernauswahlzeichen, daß ab der entsprechenden Schablonenstelle die Nullenunterdrückung aufgehoben ist, d.h. alle ab dieser Stelle erscheinenden [- oder ▽ -Zeichen werden auf jeden Fall durch eine Ziffer aus dem zweiten Operandenfeld ersetzt, wobei es gleichgültig ist, ob diese ungleich 0 oder gleich 0 ist.

Die folgenden Beispiele sollen die ED-Instruktion veranschaulichen:

Inhalte von Registern und Feldern (ausgedrückt durch Anfangs- und Endadresse der von ihnen belegten Hauptspeicherplätze):

Register 1	0000 0000
Register 2	0000 8F00
Register 3	Adresse der ersten Instruktion
008 F00 bis 008 00F	Ergebnisfeld
008 F10 bis 008 F14	Gepacktes Dezimalfeld. Wir wollen die folgenden vier Fälle betrachten: a) 00 12 34 56 7+ b) 00 00 00 81 3- c) 00 00 00 00 0+ d) 99 00 05 78 9-
008 F16 bis 008 F17	FF FF (-1 zur Basis 2 also)
008 F20 bis 008 F2F	4020 6B20 2020 6B20 2020 4B21 2040 C3D9 (Erste Schablone: b▽, ▽▽▽ , ▽▽▽ .[▽bCR)
008 F30 bis 008 F3F	4040 4040 206B 2020 206B 2020 214B 2020 (Zweite Schablone: bbbb▽,▽▽▽,▽[▽.▽▽)
008 F40 bis 008 F4F	5C40 406B 2020 206B 2020 206B 2020 2160 (Dritte Schablone: *bb,▽▽▽,▽▽▽,▽▽[-)

Instruktionen:

```
MVC     D2 0F 2 000 2 020
ED      DE 0F 2 000 2 010
```

Wir übertragen also zunächst die Schablone in das Ergebnisfeld und codieren anschließend die ED-Instruktion. Mit der ersten Schablone erhalten wir die folgenden vier Resultate:

a) bbbb12,345,67bbb
b) bbbbbbbbb8.13bCR
c) bbbbbbbbbb.00bbb
d) b9,900,057.89bCR

Verwenden wir als erste Instruktion die Instruktion

```
MVC     D2 0F 2 000 2 030 ,
```

wodurch die zweite Schablone in das Ergebnisfeld übertragen wird, so bekommen wir:

a) bbbbbbb12,345.67
b) bbbbbbbbbbbb8.13
c) bbbbbbbbbbbb0.00
d) bbbb9,900,057.89

Wir sollten zu diesen Ergebnissen bemerken, daß sich im Fall c) bei Verwendung der 2. Schablone das Resultat 0.00 ergeben hat und nicht .00 wie bei Benutzung der 2. Schablone. Das liegt daran, daß in der 2. Schablone das Startzeichen für geltende Ziffern eher erscheint als in der 1. Schablone.

Benutzen wir schließlich als erste Instruktion die Instruktion

```
MVC     D2 0F 2 000 2 040 ,
```

so wird die dritte Schablone ins Ergebnisfeld übertragen und wir erhalten damit:

a) ******1,234,567
b) **********813-
c) **********Ø
d) ****99Ø,ØØ5,789-

Die Instruktion EDMK ist identisch der ED-Instruktion; zusätzlich wird außerdem ins Register 1 die Adresse der ersten gültigen Ziffer gestellt. Diese Information kann dazu benutzt werden, um gleitende Zeichen, wie z.B. ein Dollarzeichen, ins Ergebnisfeld einzufügen. Wir wollen das an einem Beispiel veranschaulichen: (Abb. 7.21).

```
MVC     |     D2 ØF 2 ØØØ 2 Ø3Ø
EDMK    |     DF ØF 2 ØØØ 2 Ø1Ø
AH      |     4A 1 Ø 2 Ø16
BC      |     47 B Ø 3 Ø18
MVI     |     92 6Ø 1 ØØØ

Die nächste auszuführende Instruktion
besitzt die Distanzadresse Ø18
```

Abb. 7.21. Beispiel für die Instruktion EDMK

Die Instruktion AH subtrahiert 1 von der durch EDMK-Instruktion ins Register 1 gestellten Adresse. Wir könnten das gleiche auch durch die BCT- oder durch die BCTR-Instruktion erreichen. Die MVI-Instruktion wird übersprungen, wenn der zweite Operand nicht negativ ist. Die MVI-Instruktion hat zur Folge, daß in das Ergebnisfeld unmittelbar vor die erste gültige (von Null verschiedene) Ziffer ein Minuszeichen (sed 60) eingefügt wird. Damit ergeben sich folgende aufbereiteten Felder:

a) bbbbbbb12,345.67
b) bbbbbbbbbbb-8.13
c) bbbbbbbbbbbbØ.ØØ
d) bbb-9,9ØØ,Ø57.89

ED und EDMK werden als Instruktionen angesehen, die zur Dezimaleinrichtung gehören. Sie sind gleichzeitig die einzigen Instruktionen dieser Einrichtung, die nicht durch ein ungültiges Vorzeichen unterdrückt werden. Wird bei der Instruktionsdurchführung ein ungültiger Vorzeichencode entdeckt, so wird die Operation erst an dieser Stelle abgebrochen.

7.19. Die Instruktion „Ausführen”

Die EX-Instruktion ermöglicht eine besondere Art der Verzweigung.

Ausführen — Execute	44	EX	RX

Die durch den zweiten Operanden festgehaltene Adresse ist die Adresse einer Instruktion, die man als untergeordnete Instruktion zu bezeichnen pflegt, und die durch den Ablauf der EX-Instruktion ausgeführt wird. Da diese Adresse eine Instruktionsadresse darstellt, muß ihr eine Halbwortgrenze zugeordnet sein. Der erste Operand spezifiziert wie üblich ein allgemeines Register, dessen Inhalt mit Ausnahme des letzten Bytes, also der Bitstellen 24 bis 31, ohne Bedeutung ist.

Zweck der EX-Instruktion ist es, die Möglichkeit zu bieten, daß eine einzelne Instruktion, die nicht in die Instruktionsfolge hineingestellt ist, auch außerhalb der normalen Reihenfolge ausgeführt werden kann. Außer der EX-Instruktion selbst kann jede beliebige andere Instruktion auf diese Weise ausgeführt werden. Wird beim Ablauf eines Programmes die EX-Instruktion gefunden, so geht der Computer zu der Adresse über, die durch den zweiten Operanden angegeben ist und liest die dort gespeicherte Instruktion ins Steuerwerk ein. Die so abgerufene Instruktion wird in der üblichen Art ausgeführt: Der Instruktionslängenschlüssel wird stets so gesetzt, daß er die Instruktionslänge 2 anzeigt, d.h. die Länge der EX-Instruktion und nicht die Länge der untergeordneten Instruktion. Außerdem enthält das PSW als Adresse der nächsten Instruktion die Adresse derjenigen Instruktion, die im Hauptspeicher unmittelbar auf die EX-Instruktion gespeichert ist (sie wurde also aus der Adresse der EX-Instruktion durch Addition von 2 gefunden).

Bewirkt die untergeordnete Instruktion einen Sprung, so wird die Verzweigung ganz normal ausgeführt, d.h. als ob die EX-Instruktion nicht das auslösende Moment gewesen wäre. In allen anderen Fällen wird als nächste Instruktion diejenige ausgeführt, die der EX-Instruktion unmittelbar nachfolgt.

Wenn R_1 auf 0 steht, so wird die untergeordnete Instruktion genau so ausgeführt, wie wir es weiter oben beschrieben haben. Enthält jedoch R_1 einen von 0 verschiedenen Wert, so wird das letzte (rechtsbündige) Byte von R_1 durch die logische Oder-Operation mit dem zweiten Byte der untergeordneten Instruktion verknüpft, bevor dieselbe zur Ausführung kommt. Somit ist eine Modifizierung eines Teiles der untergeordneten Instruktion möglich. Bei Instruktionen im RR-Format bzw. im RX-Format spezifiziert dieses Byte Register, bei Instruktionen im SI-Format enthält dieses Byte den Direktoperanden, und bei Instruktionen im SS-Format wird durch dieses Byte bekanntlich die Länge (bzw. werden die Längen) festgehalten. Solche Instruktionen wie TRT laufen gewöhnlich unter Kontrolle der EX-Instruktion ab, wobei durch die EX-Instruktion die Länge des ersten Operandenfeldes variabel gestaltet werden kann. Wenn z.B. ein Textabschnitt auf Punkte durchsucht werden soll, so wird mit einem Längenschlüssel (L-Feld) begonnen, der einen Wert spezifiziert, der um 1 geringer ist als die Anzahl der im Abschnitt vorhandenen Zeichen. Nach Ausführung jeder TRT-Instruktion (und damit nach dem Auffinden eines Punktes) wird in das Längenfeld eine Länge gestellt, die auf die im Abschnitt verbleibende restliche Zeichenzahl bezogen ist. Die TRT-Instruktion selbst wird mit dem Längenschlüssel 0 niedergeschrieben. Die passende Länge wird immer in einem Register zur Verfügung gestellt, so daß die EX-Instruktion das Längenfeld ergänzen kann.

7.20. Unterbrechbare Instruktionen

Zwei Instruktionen im RR-Format werden nicht so ausgeführt wie die anderen Instruktionen. Diese beiden Instruktionen heißen:

a)

Übertragen lang —— Move long	0E	MVCL	RR

b)

Vergleichen logisch lang —— Compare logical long	0F	CLCL	RR

Durch die beiden Operandenadressen, d.h. durch die beiden Register R_1 bzw. R_2, ist jeweils ein Register mit gerader Ordnungsnummer zu benennen. Tatsächlich werden durch R_1 und R_2 Registerpaare (geradzahlig - ungeradzahlig) angesprochen. Die Inhalte der vier in diese Instruktionen verwickelten Register werden so interpretiert, wie es die Abb. 7.22. zeigt.

Register	Bytes			
	0	1	2	3
R_1	0	Adresse A_1 des ersten Operanden		
R_1+1	ignoriert	Länge L_1 des ersten Operanden		
R_2	0	Adresse A_2 des zweiten Operanden		
R_2+1	p	Länge L_2 des zweiten Operanden		

Anmerkung: p wird Lückenzeichen genannt

Abb. 7.22. Registerinhalte bei den Instruktionen MVCL und CLCL

Wie die durch eine solche Instruktion veranlaßte Operation abläuft, soll am Beispiel der Instruktion MVCL demonstriert werden. Folgende Einzelschritte werden nacheinander durchgeführt:

1) Die Bytes 0 der Register R_1 und R_2 werden auf 00 (sed) gesetzt.

2) Es wird L_3 als Minimum von L_1 und L_2 bestimmt.

3) Wenn eine der drei Bedingungen

   a) $A_2 < A_1 \leq A_2+L_3-1 < 16777216$

   b) $16777216-L_3+1 \leq A_2 < A_1$

   c) $A_1 \leq A_2+L_3-1-16777216$

   erfüllt ist, so wird der Bedingungsschlüssel auf 3 gesetzt und die Ausführung abgebrochen.

4) Wenn $L_1=L_2=0$ ist, dann wird der Bedingungsschlüssel auf 0 gesetzt und die Ausführung abgebrochen.

5) Wenn $L_1<L_2$ ist, dann wird der Bedingungsschlüssel auf 1 gesetzt.

6) Wenn $L_1>L_2$ ist, dann wird der Bedingungsschlüssel auf 2 gesetzt.

7) Wenn $L_1=L_2$ ist, dann wird der Bedingungsschlüssel auf 0 gesetzt.

8) Wenn $L_1=0$ ist, dann wird die Ausführung abgebrochen.

9) Wenn $L_2>0$ ist, so wird der Inhalt des Bytes auf A_2 nach A_1 übertragen. Von L_1 und L_2 wird jeweils subtrahiert sowie auf A_1 und A_2 jeweils 1 addiert. Danach wird zum Schritt 8) zurückgegangen.

10) Wenn $L_2=0$ ist, so wird das Lückenzeichen nach A_1 transportiert. Von L_1 wird 1 subtrahiert und außerdem 1 auf A_1 addiert. Danach wird zum Schritt 8) zurückgegangen.

Die Prüfungen, die im Schritt 3) vorgenommen werden, führen zu einem Abbruch der Ausführung, wenn die Operanden sich derart überlappen, daß ein Zeichen, das in ein Byte des ersten Operanden übertragen wird, im weiteren Verlauf der Instruktionsausführung als Quelle für den zweiten Operanden dient. Besitzt der Computer 16777216 Hauptspeicherplätze, so kann die Übertragung zyklisch vom Hauptspeicherplatz mit der sedezimalen Adresse FFFFFF zum Hauptspeicherplatz 000000 weitergehen. Die Schritte 8 bis 10 bilden eine Schleife. Da diese Millionen Male durchlaufen

werden kann, ist es möglich, daß die Instruktionsablaufzeit eine Sekunde anwachsen kann. Da eine solche Zeit zur Ausführung einer einzigen Instruktion als zu groß empfunden werden kann, ist eine Unterbrechung dieser Instruktion jederzeit möglich, allerdings nur während des Schrittes 8). Tritt ein solches Vorkommnis ein, so verweist die Adresse im PSW nach wie vor auf die MVCL-Instruktion, so daß deren Ausführung nach Bearbeitung der Unterbrechung wieder aufgenommen werden kann. Da die Register laufend auf den neuesten Stand gebracht werden, kann die Fortsetzung der Operation anstandslos bewerkstelligt werden. Man hat ferner bei Aufsetzung der MVCL-Instruktion noch folgendes zu beachten: Ist die MVCL-Instruktion einer EX-Instruktion untergeordnet, so darf sich das Register R_1 der EX-Instruktion nicht auf eines der vier in die Instruktion MVCL eingehenden Register beziehen. Außerdem darf sich die MVCL-Instruktion nicht innerhalb des Feldes befinden, dessen einzelne Stellen durch das erste Adreßregister, d.h. durch R_1, angesprochen werden.

Die Instruktion CLCL läuft in folgenden einzelnen Schritten ab:

1) Alle Bitstellen des Bytes 0 der Register R_1 und R_2 werden auf 0 gesetzt.

2) Wenn $L_1=L_2=0$ ist, so wird der Bedingungsschlüssel auf 0 gesetzt und die Ausführung abgebrochen.

3) Wenn $L_1 \neq 0$ und $L_2 \neq 0$ ist, so wird das durch A_1 adressierte Zeichen mit dem durch A_2 adressierten Zeichen verglichen.

   Ist $(A_1)<(A_2)$, so wird der Bedingungsschlüssel auf 1 gesetzt und die Ausführung abgebrochen. Ist dagegen $(A_1)>(A_2)$, so wird dem Bedingungsschlüssel eine 2 zugewiesen und die Ausführung beendet. Ist jedoch $(A_1)=(A_2)$, so wird eine 1 sowohl von L_1 als auch von L_2 subtrahiert sowie eine 1 auf A_1 und auf A_2 addiert. Anschließend erfolgt ein Zurückgehen auf Schritt 2).

4) Wenn $L_1=0$ ist, so wird das Lückenzeichen p mit dem Zeichen auf A_2 verglichen. Ist $p<(A_2)$, dann wird der Bedingungsschlüssel auf 1 gesetzt und die Ausführung abgebrochen. Wenn dagegen $p>(A_2)$ ist, so wird dem Bedingungsschlüssel eine 2 zugewiesen und die Ausführung beendet. Ist jedoch $p=(A_2)$, so wird 1 von L_2 subtrahiert und die Adresse A_2 um 1 erhöht. Danach erfolgt ein Zurückgehen, damit der Schritt 2) wiederholt werden kann.

5) Wenn $L_2=0$ ist, so wird das Lückenzeichen p mit dem durch A_1 adressierten Zeichen verglichen. Ist $(A_1)<p$, so wird der Bedingungsschlüssel auf 1 gesetzt und die Ausführung abgebrochen. Ist dagegen $(A_1)>p$, so wird dem Bedingungsschlüssel eine 2 zugewiesen und die Ausführung beendet. Wenn jedoch $(A_1)=p$ ist, so wird L_1 um 1 vermindert und A_1 um 1 erhöht. Danach wird zum Schritt 2) zurückgegangen.

Die Ausführung der Instruktion CLCL kann jederzeit unmittelbar nach dem Schritt 2) unterbrochen werden. Die Wiederaufnahme der Operation erfolgt in gleicher Weise wie bei der Instruktion MVCL.

Es ist zu beachten, daß nach Beendigung der durch MVCL bewirkten Operation L_1 auf 0 gesetzt ist und R_1 die um 1 erhöhte Adresse der letzten Stelle enthält, auf die ein Zeichen übertragen wurde. Das Lückenzeichen wird gewöhnlich zu 00 (sed) bzw. 40 (sed) [1] gewählt, es kann jedoch auch jedes andere Zeichen hierfür verwendet werden. Nach Abschluß der durch CLCL hervorgerufenen Operation enthalten bei einem Wert des Bedingungsschlüssels ungleich 0 (Ungleichheit) die Register R_1 und

1) Zur Erinnerung, 40 (sed) ist die Verschlüsselung des Leerzeichens im EBCDI-Code.

R_2 die Adressen derjenigen Bytes, deren Inhalte als verschieden festgestellt wurden. Der kürzere Operand wird gewissermaßen mit dem Lückenzeichen aufgefüllt, d. h. bei gleichlangen Operanden besitzt das Lückenzeichen überhaupt keine Bedeutung.

7.21. Die Instruktionsformate

Im Verlauf des Kap. 7 haben wir alle Instruktionsformate kennengelernt, soweit wir nicht schon aus früheren Kapiteln mit ihnen vertraut waren. Wir halten es für zweckmäßig, wenn wir alle Formate in einer Übersicht (Abb. 7.23.) zusammenfassen.

Aufgabengruppe 18

1) Gegeben sind folgende Inhalte:

Register 11	0000 1000
Register 10	0030 089F
Speicherplätze 000 028 bis 000 02F	
laufendes PSW	FF00 0000 0F00 0D00
	FF01 0000 0F00 1000

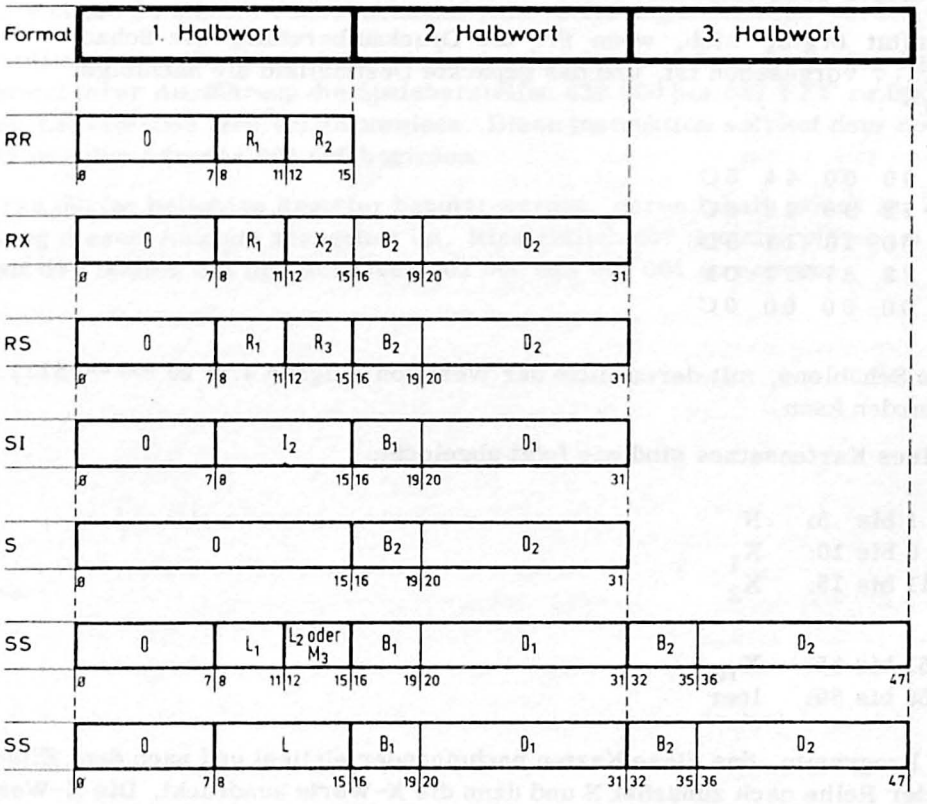


Abb. 7.23. Übersicht über die Instruktionsformate

Gesucht ist der Inhalt der Speicherplätze 000 028 bis 000 02F nach Ausführung der nachfolgenden Instruktionen, die jeweils auf dem Hauptspeicherplatz mit der sedezimalen Adresse 001 000 beginnen sollen.

a) L	58 5A 0 004
b)	00 BB
c) SRDL	8C 50 0 004
d) AP	FA 21 0 028 0 02D
e) DP	FD 21 0 02A 0 02D
f) BC	47 FB 0 007

Es soll angenommen werden, daß die betreffende IBM /370 einen Hauptspeicher mit 65 536 Bytes besitzt, d.h. die höchstmögliche Adresse ist 00FFFF (sed).

2) Angenommen, daß numerische Daten in sedezimaler Form eingegeben und in gepackte Dezimalzahlen umgewandelt werden sollen. Als Teil dieser Umwandlungsprozedur müssen die Buchstaben A bis F in die Bitmuster 1010 bis 1111 übersetzt werden. Stelle deshalb eine Tabelle zusammen, die F0 bis F9 in F0 bis F9 und C1 bis C6 (d.h. A bis F) in FA bis FF übersetzen kann.

3) Bei einem auf den Speicherplätzen 002 000 bis 002 00F gespeichertem Feld wollen wir alle Kommas und Punkte gegenseitig austauschen. Somit würde z. B. aus 2,123,456,981.14 der Feldinhalt 2.123.456.981,14 werden; wir planen also nichts anderes als eine Überführung der amerikanischen Schreibweise in die europäische Schreibweise. Baue eine entsprechende Tabelle auf und verwende die TRT- und EX-Instruktionen in einem Programm, das dieses Problem löst.

4) Welches Resultat ergibt sich, wenn für die Druckaufbereitung die Schab b▽,▽▽▽,▽▽▽.[▽ vorgesehen ist, und das gepackte Dezimalfeld die nachfolgei Werte enthält?

a)	00 00 00 44 5C
b)	00 12 98 78 6C
c)	00 00 13 11 9D
d)	22 33 87 77 0F
e)	00 00 00 00 0C

5) Entwerfe eine Schablone, mit deren Hilfe der Wert von Aufgabe 4. c) zu *******$131.19- aufbereitet werden kann.

6) Die Karten eines Kartensatzes sind wie folgt abgelocht:

Spalten 1 bis 5:	N
Spalten 6 bis 10:	X_1
Spalten 11 bis 15:	X_2
⋮	
Spalten 51 bis 55:	X_{10}
Spalten 56 bis 80:	leer

Schreibe ein Programm, das diese Karten nacheinander einliest und nach dem Einlesen jeder Karte der Reihe nach zunächst N und dann die X-Werte ausdruckt. Die X-Werte sollen dabei in der Form ±xxx.xx oder in der Form b±xx.xx oder in der Form bb±x.xx gedruckt werden, je nachdem, wie viele gültige Ziffern vor dem Dezimalpunkt stehen (auf der ersten Stelle links vom Dezimalpunkt soll auf jeden Fall eine Ziffer erscheinen, auch wenn diese gleich 0 ist). Zwischen je zwei ausgedruckten Feldern sollen drei Leerstellen auftreten.

7) Es soll ein Sachregister gedruckt werden. Die Sachbegriffe sind auf Karten in den Spalten 1 bis 71 abgelocht; die Bezugnahmen auf die einzelnen Seiten sind in den Spalten 72 bis 74, 75 bis 77 und 78 bis 80 enthalten. Es wird hierbei vorausgesetzt, daß jeder Sachbegriff auf höchstens drei Seiten erscheint. Die auf den Karten stehenden Informationen sollen in folgender Form [1] gedruckt werden:

```
BANDSPEICHER. . . . . b7,b189,b245
BANDSPULE. . . . . 289
BASIS. . . . . b124,b135,b237
```

Die gesamte Zeilenbreite beträgt in jedem Fall 47 Druckstellen. Vor jeder Seitennummer soll ein Leerzeichen erscheinen, zwischen zwei Seitennummern zusätzlich ein Komma. Die übrigen Stellen der Zeile sollen, wie oben gezeigt, durch Punkte ausgefüllt werden.

Für das obige erste Beispiel müßte die Karte wie folgt abgelocht werden:

Spalten 1 bis 12:	BANDSPEICHER
Spalten 13 bis 71:	leer
Spalten 72 bis 80:	007189245

Schreibe ein Programm, das diese Karten liest und das das Sachregister ausdruckt [2].

8) Erweitere die Aufgabe 7) so, daß auf eine Seite genau 48 Zeilen gedruckt werden. Außerdem soll die Seitenzahl rechts oben auf jeder Seite angeschrieben werden.

9) Es soll eine einzelne Instruktion niedergeschrieben werden, die in der Lage ist, während ihrer Ausführung die Speicherstellen 038 000 bis 047 FFF zu leeren, d.h. ihnen Leerzeichen (sed 40) zuzuweisen. Diese Instruktion soll auf dem Speicherplatz mit der Adresse 002 000 beginnen.

Hierzu dürfen beliebige Register benutzt werden, deren Inhalt jedoch im Zuge der Lösung dieser Aufgabe anzugeben ist. Hinsichtlich der Speicherplätze ist nur der Inhalt der Stellen mit den Adressen 002 000 und 002 001 zu nennen.

---

1) b bedeutet hier wie üblich das Leerzeichen.

2) Die Sachbegriffe sind im Bedarfsfall auf höchstens 33 Zeichen abzukürzen.

8. Das virtuelle Hauptspeicherkonzept

8.1. Das Datenverarbeitungssystem IBM /360

Bis zum Jahre 1951 wies jeder konstruierte Computer eine Instruktionsmenge auf, die sich vollständig von der jedes anderen Computers unterschied. Genauso unterschiedlich waren die Schnittstellen zu den Kanälen. Die Schlußfolgerung, daß bis dahin keine zwei Computer mit den gleichen EA-Geräten ausgerüstet werden konnten und daß die für einen Computer geschriebenen Programme nicht auf einen anderen gelegt werden konnten, ist leicht zu ziehen, ist evident.

Im Jahre 1951 erschien der Computer UNIVAC I. Bevor er sechs Jahre später durch einen verbesserten Typ abgelöst wurde, waren von ihm 32 Einheiten ausgeliefert worden. Als der UNIVAC II auf dem Markt auftauchte, konnten auf diesem Typ auch Programme ablaufen, die für den UNIVAC I geschrieben waren. Man konnte allerdings für den UNIVAC II Programme schreiben, die nicht auf dem UNIVAC I ausgeführt werden konnten. Man sagt, daß der UNIVAC II aufwärts verträglich (aufwärts kompatibel) zum UNIVAC I war; er war, wie wir soeben angedeutet haben, jedoch nicht abwärts verträglich zum UNIVAC I.

Während dieser Zeitperiode konnte bezüglich der Geräteausstattung das gleiche auch von anderen Herstellern gesagt werden. Zwischen einigen IBM-Computertypen konnten außerdem Magnetbandlaufwerke ausgetauscht werden, jedoch nicht die Steuereinheiten.

Im Jahre 1964 führte die IBM ihr System IBM /360 ein. Das Datenverarbeitungssystem IBM /360 wurde von vornherein als Computerfamilie entworfen; alle ihre Mitglieder weisen die gleiche Maschinensprache auf. Zur Familie gehören im einzelnen die Modelle 20, 22, 25, 30, 40, 44, 50, 65, 67, 75, 85, 91 und 95; unter diesen besitzen die Modelle 20, 44, 67, 91 und 95 einige spezifische Besonderheiten. So ist z.B. das Modell 67 grundsätzlich gleich den anderen Modellen, es weist jedoch zusätzlich besondere Maschineninstruktionen auf, die die dynamische Adreßumsetzung einschließt. Diese Einrichtung wurde ihres wesentlichen Vorteils wegen in der Folge für alle Modelle der Computerfamilie IBM /370 kopiert. Sie wird eine der Errungenschaften der Datenverarbeitungstechnik sein, die wir in diesem Kapitel kennenlernen werden. Die beiden Modelle 91 und 95 unterscheiden sich vom engeren Familienkreis nur unwesentlich: Der Hauptunterschied zeigt sich in der geringfügig anderen Behandlung einiger Programmunterbrechungen. Beim Modell 44 fehlen einige Instruktionen, so z.B. alle Instruktionen des SS-Formats. Eine Zusatzeinrichtung wurde einige Jahre nach Installation der ersten 44er Modelle auf den Markt gebracht, die die fehlenden Instruktionen beinhaltete. Einige Maschinen wurden damit auch ausgerüstet. Das Modell 20 unterscheidet sich wesentlich von den Hauptvertretern der Familie IBM /360. Bei ihm wurde einmal nur ein kleiner Teil der vollen Instruktionsmenge eingebaut und zum anderern besitzt es andere Kanalschnittstellen als die größeren Modelle. Es war in erster Linie für die Realisiertung kleiner Abrechnungssysteme gedacht. Nahezu 30000 Systeme IBM /360 wurden produziert, eingeschlossen in dieser Zahl sind über 10000 Modelle 20.

Der wesentliche Unterschied zwischen den Modellen der Familie IBM /360 besteht einmal in der Hauptspeicherkapazität und zum anderen in der internen Verarbeitungsgeschwindigkeit. Beispielsweise schwankt die Ausführungszeit für eine Additionsinstruktion im RX-Format zwischen 50μs beim Modell 25 und nur 0,06μs beim Modell 95. Die geringste Hauptspeicherkapazität beträgt beim Modell 30 gerade 8192 Bytes,

die maximale beim Modell 95 immerhin 5242880 Bytes. Aber, abgesehen von einigen Einschränkungen, können alle Programme zwischen den verschiedenen Modellen ausgetauscht werden; eine Ausnahme hiervon bildet lediglich das Modell 20. Zu den eben genannten Einschränkungen sind unter anderen folgende zu zählen: a) Benutzung modellabhängiger Einrichtungen, wie z.B. der Instruktion „Diagnose", b) Codierung zeitabhängiger Programme, die z.B. so vorgehen, daß sie eine SIO-Instruktion hinausschicken und nach einigen hundert Instruktionen, d.h. nach Ablauf einer bestimmten Instruktionsmenge, versuchen, Daten im Eingabegebiet zu verarbeiten, ohne vorher zu prüfen, ob die durch SIO eingeleitete Operation auch tatsächlich abgeschlossen ist, und c) ungenügende Anzahl von Hauptspeicherstellen bei einem Modell.

8.2. Das Datenverarbeitungssystem IBM /370

Obwohl die Modelle der Computerfamilie IBM /360 aufwärts und abwärts verträglich sind, war die Abwärtsverträglichkeit doch wegen der begrenzten Hauptspeicherkapazitäten stark eingeschränkt. Bei Digitalrechnern überdauert eine Maschinengeneration nach den bisherigen Erfahrungen eine Zeitspanne von rund sechs Jahren. Danach ist der technische und technologische Fortschritt so weit gediehen, daß eine fortdauernde Produktion zu einem unwirtschaftlichen Prozeß ausartet. Als es 1970 Zeit wurde, die IBM /360 durch Nachfolgemodelle abzulösen, entschied sich die IBM für die Produktion einer Modellreihe, die aufwärts verträglich zur IBM /360 sein sollte. U.a. wurden folgende neue Einrichtungen ausgewählt, um die Eigenschaften der IBM /360 sinnvoll zu erweitern:

a) Hinzufügen neuer Instruktionen, wie z.B. MVCL, ICM und SRP.
b) Einbau der Steuerregister
c) Einbau der Blockmultiplexkanäle

Die neue Computerfamilie wurde IBM /370 genannt. Die ersten ausgelieferten Modelle dieser Familie waren im Grunde genommen nur verbesserte Modelle der IBM /360 mit den erwähnten zusätzlichen Einrichtungen. Wir meinen hiermit speziell folgende Modelle der IBM /370: 155-1, 155-3, 155-4, 165-1 und 195.

Während der beiden Jahre 1971 und 1972 ergänzte die IBM die Computerfamilie IBM /370 durch neue Modelle und neue Ausrüstungen. Die hauptsächlichste neue Einrichtung war die dynamische Adreßumsetzung , eine weiterentwickelte Version der schon bei der IBM /360, Modell 67 vorhandenen Einrichtung, die als äußerst wertvoll gefunden wurde. Diese Einrichtung erfordert jedoch das Programmstatuswort (PSW) für die erweiterte Steuerung. Die frühen Modelle der Computerfamilie IBM /370 arbeiteten ausschließlich mit der Basissteuerung. Sicher wird auch einmal die IBM /370 wieder durch einen neuen Computertyp ersetzt werden. Doch vorerst kann sie als Nonplusultra der Datenverarbeitung gelten.

In diesem Buch legen wir unser Hauptaugenmerk auf die Modelle, die mit der erweiterten Steuerung arbeiten. Abgesehen von einigen frühen, weiter oben erwähnten Modellen der Familie IBM /370 wird die auch für das System IBM /370 zur Verfügung stehende Basissteuerung hauptsächlich für laufende Programme benutzt, die für die IBM /360 entwickelt wurden.

8.3. Virtuelle Adressen

Bei der Computerfamilie IBM /360 war das Hauptproblem, das beim Programmaustausch aufkam, darin zu sehen, daß bei den kleineren Maschinen nicht genügend Hauptspeicherplatz zur Verfügung stand.

Aus dieser Problematik heraus wurde die Idee des virtuellen Speichers und die des dynamischen Speichermanagements geboren. Zwei Fakten liegen dieser Idee zugrunde:

1) Die Tatsache, daß die bisherigen Maschinen und Betriebssysteme keine optimale Nutzung des Hauptspeichers ermöglichten, war nicht länger wegzuleugnen.

2) Die Überlegung, daß es nicht unbedingt erforderlich ist, ein Programm während seiner gesamten Laufzeit in seiner vollen Ausdehnung im Hauptspeicher zu halten, basiert auf den bisher gemachten praktischen Erfahrungen. Es muß vielmehr ausreichen, wenn nur diejenigen Teile eines Programmes im Hauptspeicher liegen, die gerade für die Ausführung herangezogen werden müssen. Die inaktiven Programmteile können bis zu ihrer Aktivierung durchaus auch auf externen Speichern untergebracht sein.

Da sich die Definition eines Programmteiles als aktiv oder inaktiv nicht grundsätzlich, sondern immer nur auf einen bestimmten Zeitpunkt bezogen treffen läßt, d.h. inaktive Programmteile werden im Verlauf der Programmausführung zu aktiven und umgekehrt, erfordert eine solche Überlegung logischerweise den Austausch von Programmteilen zwischen dem Hauptspeicher und dem für die Programmspeicherung vorgesehenen Externspeicher [1].

Ein solches Auswechseln, das ja einem EA-Vorgang mit dem hierfür erforderlichen üblichen Aufwand entspricht, geschieht sinnvollerweise nicht byte- oder instruktionsweise, sondern in 4096 oder 2048 Bytes großen Blöcken, die Seiten genannt werden. Ein Programm wird also zur Ausführung seitenweise in den Hauptspeicher gebracht.

Zur Lösung unseres Hauptspeicherproblems kann die eben in den Umrissen skizzierte Auslagerung von Hauptspeicherinhalten auf externe Speichergeräte sinnvoll beitragen. Im Hauptspeicher werden also nur die gerade benutzten Seiten gehalten. Jede Instruktion geht nach ihrer Entschlüsselung durch einen dynamischen Adreßumsetzer [2], eine Hardware-Standardeinrichtung. Diese sieht in Adreßumsetzungstabellen nach, die im unteren Teil des Hauptspeichers liegen. Befindet sich die Seite bzw. befinden sich die Seiten im Hauptspeicher, die die von den Instruktionen gebildeten Adressen enthalten, so werden die Adreßstellen höherer Ordnung mit der Adresse versorgt, auf der die entsprechende Seite bzw. entsprechenden Seiten beginnen. Befindet sich im Zugriffsaugenblick eine Seite, auf die bezug genommen wird, nicht im Hauptspeicher, so kommt es zu einer Programmunterbrechung, die zu einer Routine verzweigt, die dafür sorgt, daß die erforderliche Seite in den Hauptspeicher gebracht wird, auf diese Weise eine andere Seite überlagernd. Danach kann die Ausführung der Instruktion erneut versucht werden.

Eine solche Adreßumsetzungsoperation nimmt natürlich eine gewisse Zeit in Anspruch, wodurch die Entschlüsselung einer Instruktion verlangsamt wird. Befindet sich die angeforderte Seite nicht im Hauptspeicher, so muß das Programm auf ihre Transferierung warten und wird dadurch wesentlich verzögert. Nichtsdestotrotz ermöglicht es eine solche Methode, externe Speicherplätze wie Hauptspeicherplätze zu benutzen.

---

1) Meist werden wohl hierzu Magnetplattenspeichergeräte eingesetzt.

2) Im angloamerikanischen Sprachraum als „dynamic address translator", abgekürzt DAT, bezeichnet.

Da externe Speicherstellen beträchtlich billiger als Hauptspeicherstellen sind, sind die meisten Installationen zwar mit einer beschränkten Hauptspeicherkapazität, jedoch mit einer dieser gegenüber praktisch unbegrenzten Externspeicherkapazität ausgerüstet.

Der Hauptspeicher und der für die Seitenspeicherung vorgesehene Teil des externen Speichers heißen zusammen virtueller Speicher. Der zur Verfügung stehende Hauptspeicher selbst wird in diesem Zusammenhang als Realspeicher bezeichnet. Wir sprechen von einer logischen oder virtuellen Adresse, wenn wir eine Adresse im virtuellen Speicher meinen. Unter einer realen Adresse verstehen wir die Adresse eines Hauptspeicherplatzes nach erfolgter Adreßumsetzung.

Je größer das Verhältnis zwischen dem von einem Programm belegten virtuellen Speicherplatz und dem für dieses Programm zur Verfügung stehenden realen Speicherplatz ist, um so mehr muß mit Seitenfehlbedingungen gerechnet werden, d.h. mit Bezugnahmen auf Seiten, die sich zur Ansprechzeit gerade nicht im Realspeicher befinden. Jeder solche Seitenfehler erfordert einen „Ausflug" auf den externen Speicher, um eine neue Seite in den Hauptspeicher herüberzuholen; in Konsequenz davon reduziert sich die Programmablaufgeschwindigkeit. Beim gegenwärtigen Entwicklungsstand der elektronischen Technik kam jedoch die IBM zu dem Schluß, daß es billiger sei, schnellere Computer herzustellen als Computer mit riesigen Hauptspeichern auszurüsten. Um einmal eine Zeit zu nennen: Es ist nicht ungewöhnlich, daß eine IBM /370 den vierten Teil ihrer Betriebszeit dem laufenden Seitenwechsel spenden muß. Wenn jedoch ein Computer schnell genug ist, so kann er dennoch die für ihn vorgesehene Arbeit ausführen.

8.4. Dynamische Adreßumsetzung

Wir wollen zunächst einige grundlegende Aussagen machen.

1) Das Steuerregister 0 enthält auf den Bitstellen 8 und 9 die Seitengröße in folgender Weise:

   01 ⟶ 2k≙2048 Bytes
   10 ⟶ 4k≙4096 Bytes

2) Das Steuerregister 0 enthält auf den Bitstellen 11 und 12 die Segmentgröße in folgender Weise:

   00 ⟶ 64k≙ 65536 Bytes
   10 ⟶ 1024k≙1048576 Bytes

3) Das Steuerregister 0 enthält auf der Bitstelle 10 die Größe einer Seiteneintragung in folgender Weise:

   0 ⟶ 2 Bytes
   1 ⟶ 4 Bytes

   Für unsere Zwecke muß auf dieser Bitstelle immer eine 0 stehen.

4) Die Bitstellen 0 bis 7 des Steuerregisters 1 spezifizieren die Länge der Segmenttabelle. Der durch sie verifizierte, um 1 vermehrte und mit 64 multiplizierte Wert gibt die Länge der Segmenttabelle an. Als maximalmögliche Länge folgt daraus $256 \cdot 64 = 16384$ Bytes.

5) Die Bitstellen 8 bis 25 des Steuerregisters 1 enthalten die verkürzte Adresse der Segmenttabelle. Die Adresse selbst wird durch Multiplikation des Inhalts dieser Bitstellen mit 64 gebildet.

Die effektiven Adressen, die aus den Adreßteilen der Instruktionen gebildet werden, sind virtuelle Adressen. Sie werden so zerlegt, wie es die Abb. 8.1. zeigt.

Bitstellen	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25	26 27 28 29 30 31	32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
Inhalt	ohne Bedeutung	Segment-nummer	Seiten-nummer	Byteadresse

Abb. 8.1. Bestandteile virtueller Adressen bei bestimmten Segment- und Seitengrößen

Die Segmentfolgenummer [1] beginnt stets auf der Bitstelle 8, die Seitenfolgenummer [2] auf Bitstelle 12 bei einer Segmentgröße von 1024k und auf Bitstelle 16 bei einer Segmentgröße von 64k. Die Byteadresse fängt auf der Bitstelle 20 an, wenn die Seitengröße 4k beträgt, bei einer 2k großen Seite dagegen erst bei Bitstelle 21. In Abb. 8.1. ist die Segmentgröße zu 64k und die Seitengröße zu 2k gewählt.

Um zu einer realen Adresse zu kommen, müssen nach der Bildung der effektiven Adresse drei weitere logische Schritte vollzogen werden. Den ersten Schritt zur Adreßumsetzung haben wir in Abb. 8.2. dargestellt; er zeigt uns eine Addition zweier Größen zwecks Bildung einer aus 24 Bits bestehenden Adresse.

Bitstellen des Ergebnisses	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
1. Summand	Bitstellen 8 bis 25 des Steuerregisters 1 | 0 0 0 0 0 0
2. Summand	0 0 0 0 0 0 0 0 0 0 0 0 0 0 Segmentnummer | 0 0
Ergebnis	Summe aus Segmentnummer und Adresse der Segmenttabelle | 0 0

Abb. 8.2. Ermittlung der realen Adresse (1. Schritt)

Wenn die entstehende 24 Bitstellen umfassende Adresse nicht innerhalb der Segmenttabelle liegt, was durch Heranziehung der durch die Bitstellen 0 bis 7 des Steuerregisters 1 spezifizierten Segmenttabellenlänge zu einem Vergleich erreicht wird, so tritt eine Programmunterbrechung auf (Segmentumsetzungs-Ausnahmebedingung mit dem sedezimalen Unterbrechungsschlüssel 0010). Liegt die errechnete Adresse innerhalb der Segmenttabelle, so verweist sie auf eine in ihr enthaltene, ein Wort lange Eintragung, die die in Abb. 8.3. bildlich dargestellten Informationen enthält.

Bitstellen	0 1 2 3	4 5 6 7	8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	32 33	34 35
Inhalt	Längen-schlüssel	0 0 0 0	Anfangsadresse der Seitentabelle	0 0	G

Abb. 8.3. Eintragung in der Segmenttabelle

1) Kurz auch: Segmentnummer
2) Kurz auch: Seitennummer

Die gefundene Eintragung in der Segmenttabelle ermöglicht den Zugriff zur Seitentabelle. Voraussetzung zu diesem zweiten Schritt der dynamischen Adreßumsetzung ist, daß die Gültigkeitsbitstelle G auf 0 steht; eine 1 zeigt an, daß eine Segmentumsetzungs-Ausnahmebedingung vorliegt. In der Eintragung der Segmenttabelle finden wir also die Anfangsadresse der Seitentabelle. Ihre wahre Länge wird aus dem Längenschlüssel ermittelt, auf den zunächst eine 1 addiert werden muß. Anschließend ist die entstandene Summe mit einem Faktor f zu multiplizieren. Dabei wird f so angesetzt, wie es aus der Übersicht von Abb. 8.4. zu entnehmen ist.

Seitengröße	Segmentgröße	Multiplikator
4k	64k	2
2k	64k	4
4k	1024k	32
2k	1024k	64

Abb. 8.4. Multiplikatoren zur Bestimmung der Länge der Seitentabelle

Die vier Bits höchster Ordnung der Seitennummer werden mit dem Inhalt der Bitstellen 0 bis 3 der Eintragung in der Segmenttabelle verglichen. Ist die Seitennummer größer, so tritt eine Programmunterbrechung auf (Seitenumsetzungs-Ausnahmebedingung mit dem sedezimalen Unterbrechungsschlüssel 0011). Anderenfalls wird die Adresse einer ein halbes Wort langen Eintragung in der Seitentabelle mittels einer Addition gefunden (Abb. 8.5.).

Bitstellen des Ergebnisses	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
1. Summand	Anfangsadresse der Seitentabelle | 0 0 0
2. Summand	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | Seitennummer | 0
Ergebnis	Summe aus Seitennummer und Adresse der Seitentabelle | 0

Abb. 8.5. Ermittlung der realen Adresse (2. Schritt)

Das Ergebnis der Addition führt uns also auf eine Eintragung in der Seitentabelle, deren Inhalt in der Abb. 8.6. beschrieben ist.

Bitstellen		0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Inhalt bei einer Seitengröße von	2k	Rahmennummer der Seite | G | 0 0 0
	4k	Rahmennummer der Seite | G | 0 0 0

Abb. 8.6. Eintragung in der Seitentabelle

Die Durchführung des nächsten Schrittes zur Ermittlung der realen Adresse hängt vom Inhalt der Gültigkeitsbitstelle G ab. Ist diese auf 1 gesetzt, so ist eine Seitenumsetzungs-Ausnahmebedingung aufgetreten. Im anderen Fall kann im 3. logischen Schritt der

Adreßumsetzung schließlich die reale Adresse geformt werden, indem an die 12 oder 13 Stellen umfassende Rahmennummer der Seite die 12 oder 11 Bitstellen umfassende Byteadresse (siehe Abb. 8.1.) angehängt wird (Abb. 8.7.).

Bitstellen	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Reale Adresse	Rahmennummer | Byteadresse
	Rahmennummer | Byteadresse

Abb. 8.7. Ermittlung der realen Adresse (3. Schritt)

Eine Programmunterbrechung mit dem sedezimalen Unterbrechungsschlüssel 0012 tritt auf, wenn eine Umsetzungsspezifikations-Ausnahmebedingung entdeckt wird.

Eine solche macht sich in folgenden Fällen bemerkbar:

a) Die Bitstellen 8 und 9 des Steuerregisters 0 enthalten entweder 00 oder 11

b) Die Bitstellen 8 und 9 des Steuerregisters 0 enthalten 10, obwohl die 4k Bytes umfassende Seiteneinrichtung nicht installiert ist

c) Die Bitstellen 11 und 12 des Steuerregisters 0 enthalten entweder 01 oder 11

d) Die Bitstellen 11 und 12 des Steuerregisters 0 enthalten 10, obwohl die die Byte [1] umfassende Segmenteinrichtung nicht installiert ist.

e) Die Bitstelle 10 des Steuerregisters 0 enthält eine 1, obwohl die 31-Bit-Adressierung nicht installiert ist.

Außerdem tritt diese Ausnahmebedingung auch dann auf, wenn rechts von der Ungültigkeitsbitstelle U einer gültigen Seitentabelleneintragung nicht lauter Nullen vorhanden sind. Bei einigen Modellen der Computerfamilie IBM /370 wird diese Bedingung zusätzlich auch gesetzt, wenn die Bitstellen 4 bis 7 bzw. 29 und 30 einer gültigen Segmenttabelleneintragung nicht sämtlich auf 0 stehen.

Beim Auftreten einer Umsetzungsspezifikations-Ausnahmebedingung wird die betreffende Operation unterdrückt. Beim Auftreten der beiden anderen Ausnahmebedingungen werden die fraglichen Operationen zunichte gemacht, d.h. daß das Programmstatuswort (PSW) auf die Adresse derjenigen Instruktion gesetzt wird, die das Auftreten der Ausnahmebedingung verursachte. Das Zustandekommen einer Segmentumsetzungs-Ausnahmebedingung deutet häufig auf einen gravierenden Programmierungsfehler, das einer Seitenumsetzungs-Ausnahmebedingung jedoch verweist meist lediglich auf die Tatsache, daß sich die erforderliche Seite nicht im realen Speicher befindet. In einem solchen Fall muß die Unterbrechungsbehandlungsroutine dafür sorgen, daß die fragliche Seite in den Hauptspeicher gebracht wird. Danach kann das alte PSW wieder geladen werden.

Halten wir zum Schluß dieses Abschnittes den Kernpunkt unserer Betrachtungen unumstößlich fest: Die virtuelle Adresse ist gleich der Summe aus den Inhalten des Basis- und evtl. des Indexregisters sowie der Distanzadresse. Von der dynamischen Adreßumsetzung wird diese Adresse in die Segment- und Seitennummer sowie in die Byteadresse aufgerissen.

1) 1M Byte (1 Megabyte) entspricht 1 048576 Bytes.

8.5. Der Austausch von Seiten (Seitenwechsel)

Einige Seiten werden auf Dauer im realen Speicher festgehalten. Zu diesen gehören beispielsweise die Seiten, die die Segmenttabellen beinhalten. Einige Seiten müssen auf eine bestimmte Zeit hin im Hauptspeicher stehenbleiben, denn die EA-Operationen z.B. können nur mit dem realen Speicher ausgeführt werden. Infolgedessen müssen alle für EA-Operationen erforderlichen Seiten vom Beginn bis zum Ende der EA-Operation im Hauptspeicher fixiert bleiben. Alle anderen Seiten brauchen sich nur zeitweise im Hauptspeicher zu befinden, sie können im Bedarfsfall von anderen Seiten überlagert werden. Bevor jedoch eine Seite überlagert werden kann, muß entschieden werden, ob die betreffende Seite während ihrer Hauptspeicherexistenz geändert wurde, d.h. z.B., daß in einige ihrer Stellen neue Werte gespeichert wurden. Wenn dem so ist, so ist natürlich die Kopie dieser Seite auf dem Externspeicher nicht mehr auf dem neuesten Stand. Deshalb muß eine solche Seite vor ihrer Überlagerung zuerst auf den Externspeicher zurückgeschrieben werden. Erfolgte jedoch keine Einspeicherung in die Seite, die überlagert werden soll, und befindet sich überdies eine Kopie dieser Seite auf dem Externspeicher, so kann sie ohne großes Getue anstandslos und sofort überschrieben werden. In jedem Fall muß jedoch ihre Eintragung in die Seitentabelle als nicht verwendbar markiert werden, um ihre Verwendung zu verhindern.

Die Instruktion „Einfügen Speicherschlüssel" mit dem mnemonischen Operationsschlüssel ISK versetzt das Programm in die Lage, nicht nur den Speicherschlüssel zu holen, sondern ebenfalls die Referenz- und Änderungsbits des betreffenden Speicherblockes. Enthält die Änderungsbitstelle eine 1, so erfolgte in den Block eine Einspeicherung, anderenfalls nicht. Auf dieser Basis kann somit entschieden werden, ob in eine zu überlagernde Seite eingespeichert wurde oder nicht.

Im Hauptspeicher können sich Seiten befinden, die gerade nicht benutzt werden. So etwas ereignet sich beispielsweise dann, wenn ein Programm abgeschlossen wurde und ein nachgeladenes neues Programm nicht alle Plätze des vorhergehenden Programmes benötigt. Augenscheinlich kann die Durchsatzleistung gesteigert werden, wenn zum Zeitpunkt einer Überlagerungsnotwendigkeit festgestellt werden kann, ob eine inaktive Seite im Hauptspeicher liegt. Zu diesem Zweck wird durch eine charakteristische Seitenunterbrechungsroutine eine Seitentabelle verwaltet, in der die in einer bestimmten Weise gemessenen Aktivitäten der einzelnen Seiten verzeichnet sind. Diese Routine kann u.a. eine Liste von inaktiven Seiten einschließen. Ist diese erschöpft, so kann die Routine die Referenzbitstellen der Seiten prüfen, um solche Seiten als inaktiv zu kennzeichnen, die zuletzt nicht benutzt wurden. Wurde eine solche Seite geändert, so muß sie auf den Externspeicher gebracht werden, bevor sie überlagert werden kann. Der von diesem Seitensupervisor benutzte Algorithmus bestimmt im wesentlichen, in welchem Maße eine Leistungssteigerung erzielt werden kann, wenn die Notwendigkeit eines Seitenaustausches aufkommt. Es sollte jedoch an dieser Stelle ausdrücklich betont werden, daß es niemals einen Algorithmus geben kann, der für alle Situationen die besten Resultate zeitigt.

Es gibt Programme, die eine gewisse Zeitabhängigkeit einschließen. Ein solches könnte z.B. Daten von einer Datenendstation (Terminal) empfangen, die es nach Ausführung einiger Kanalbefehlswörter (CCW) sofort verarbeiten muß. Ein solches Programm ist nicht in der Lage, die infolge eines Seitenwechsels eintretende Zeitverzögerung zu tolerieren. Deshalb müssen solche Programme zeitweise im Hauptspeicher festgehalten werden und alle ihre Seiten sind während der Programmablaufzeit als nicht austauschbar zu markieren.

8.6. Der Umsetzungsablagepuffer

Das Durchsuchen der Segmenttabelle und der Seitentabelle erfolgt relativ langsam. Aus diesem Grunde werden die Informationen über die jüngsten Segment-Seiten-Umsetzungen in einem besonderen Hardwarespeicher, dem Umsetzungsablagepuffer oder Adreßumsetzungsspeicher, festgehalten. Bei diesem kann es sich um einen Assoziationsspeicher [1] handeln, d.h. um einen Speicher, in dem jede Eintragung gleichzeitig nach einem vorgegebenen Argument durchsucht wird. Ein solcher Speicher ist recht kostspielig und nimmt typischerweise gegen acht Eintragungen auf. Als Suchargument dient der die Segmentnummer und die Seitennummer enthaltende Teil der effektiven Adresse (siehe Abb. 8.1.). Wird eine Übereinstimmung zwischen dem Suchargument und einer Eintragung gefunden, so tritt der Funktionswert aus dem Speicher an die Stelle der Segment- und der Seitennummer; das reguläre Durchsuchen der Segment- und der Seitentabelle (siehe hierzu Unterabschnitt 8.4.) wird dann übergangen.

Der Adreßumsetzungsspeicher [2] ist bei den einzelnen Modellen der Computerfamilie IBM /370 auf verschiedene Weisen verwirklicht worden. Generell muß jedoch betont werden, daß seine Wirkungsweise dennoch die gleiche ist. Der Adreßumsetzungsspeicher ist mit einem fest verdrahteten Algorithmus verbunden, der dafür sorgt, daß nur die jüngsten und am häufigsten angeforderten Tabelleneintragungen im Puffer bereitstehen.

Wenn durch die Seitenverwaltung eine Seite aus der Seitentabelle entfernt wird, so besteht selbstverständlich die Möglichkeit, daß der Umsetzungsablagepuffer noch eine Eintragung über diese Seite aufweist. Infolgedessen muß eine besondere Instruktion hinausgeschickt werden, die alle Eintragungen immer dann als ungültig kennzeichnet, wenn eine Seite ausgetauscht wird.

8.7. Die Instruktionen zum Ansprechen der dynamischen Adreßumsetzung

Fünf Instruktionen sind als Bestandteil der der dynamischen Adreßumsetzung dienenden Maschineneinrichtung vorhanden. Es handelt sich um folgende Instruktionen:

1)	Speichern mit nachfolgender Und-Verknüpfung der Systemmaske — Store then And System Mask	AC	STNSM	SI
2)	Speichern mit nachfolgender Oder-Verknüpfung der Systemmaske — Store then Or System Mask	AD	STOSM	SI
3)	Freimachen Umsetzungsablagepuffer — Purge TLB	B20D	PTLB	S
4)	Zurücksetzen Aufrufbit — Reset Reference Bit	B213	RRB	S
5)	Laden Realadresse — Load Real Adress	B1	LRA	RX

1) Bei einem Assozialtionsspeicher handelt es sich um einen Speicher, dessen Speicherelemente durch Angabe ihres Inhalts oder eines Teils davon aufrufbar sind.

2) Im angloamerikanischen Sprachraum als „Translation Lookaside Buffer", abgekürzt TLB, bezeichnet.

Durch die von der Instruktion STNSM ausgelöste Operation wird zunächst der Inhalt der Bitstellen 0 bis 7 des laufenden Programmstatuswortes (PSW) auf den Speicherplatz mit der effektiven Adresse E_1 gespeichert. Anschließend wird der Direktoperand I_2 durch die logische Undoperation mit dem Inhalt der Bitstellen 0 bis 7 des laufenden PSW verknüpft. Das Ergebnis dieser Verknüpfung ersetzt den Inhalt dieser Bitstellen. Diese Instruktion erweist sich als nützlich, wenn es darum geht, kritische Bitstellen des PSW auf Null zu setzen, doch gleichzeitig ihren Inhalt zu retten. Die durch die Instruktion STOSM veranlaßte Operation läuft in ähnlicher Weise ab: bei der in sie eingeschlossenen logischen Operation handelt es sich jedoch um die Oderverknüpfung. Mit dieser Instruktion werden in der Regel bestimmte Bitstellen mit einer 1 versorgt.

Die Instruktion PTLB veranlaßt, daß alle Eintragungen im TLB als ungültig markiert werden (siehe hierzu Abschnitt 8.6.). Die Operandenadresse wird bei dieser Instruktion nicht entschlüsselt und auch nicht in anderer Weise benutzt.

Durch die Instruktion RRB wird die Aufrufbitstelle (Referenzbitstelle) des Hauptspeicherblockes auf Null gesetzt, der durch die Adresse E_2 angesprochen wird. Durch diese Instruktion erfolgt also die Bezugnahme auf die reale Adresse (nicht auf die virtuelle Adresse) eines 2048 Bytes umfassenden Hauptspeicherblockes. Die Änderungsbitstelle wird von dieser Instruktion nicht berührt. Der Bedingungsschlüssel wird in Abhängigkeit von den früheren beiden Bitstellen zugewiesenen Werten gesetzt (Abb. 8.8.).

Aufrufbitstelle	Änderungsbitstelle	Bedingungsschlüssel
0	0	0
0	1	1
1	0	2
1	1	3

Abb. 8.8. Setzen des Bedingungsschlüssels bei der Instruktion RRB

Die Instruktion LRA entspricht im Konzept der Instruktion LA. Es erfolgt jedoch eine dynamische Adreßumsetzung der durch X_2, B_2 und D_2 gebildeten virtuellen Adresse in eine reale Adresse, wobei es gleichgültig ist, ob im PSW die Adreßumsetzung spezifiziert ist oder nicht. Der Bedingungsschlüssel wird wie folgt gesetzt:

0 ← erfolgreiche Adreßumsetzung
1 ← Segmenttabellen-Eintragung ungültig
(die Gültigkeitsbitstelle enthält eine 1)
2 ← Seitentabellen-Eintragung ungültig
(die Gültigkeitsbitstelle enthält eine 1)
3 ← Überschreitung der Länge der Segment- oder der Seitentabelle

Ist der Bedingungsschlüssel ungleich 0, so wird die Ausführung dieser Instruktion unterdrückt. Diese Instruktion wird nutzbringend zum Setzen von Adressen in Kanalbefehlswörtern bzw. in indirekten Adreßwörtern eingesetzt, da solche Adressen in realer Form vorliegen müssen. Die Instruktion LRA besitzt zusätzlich die Eigenschaft, daß der Umsetzungsablagepuffer (siehe Abschnitt 8.6.) unbeachtet bleibt. Somit kann sie auch zur Prüfung der Segment- und Seitentabellen herangezogen werden, sogar dann, wenn der Computer mit der Basissteuerung gefahren wird.

8.8. Die Identifizierung von Zentraleinheiten

Die privilegierte Instruktion

Speichern Identifizierung ZE —— Store CPU Identification	STIDP	B202	S

[1]

muß sich auf einen Doppelwortoperanden, der auf eine Doppelwortgrenze ausgerichtet ist, beziehen. Durch die Ausführung dieser Instruktion werden in das adressierte Doppelwort die folgenden Informationen gespeichert:

Byte 0:	Versionsnummer (zwei sedezimale Ziffern)
Bytes 1 bis 3:	Identifizierungsnummer der Zentraleinheit (sechs sedezimale Ziffern)
Bytes 4 und 5:	Modellnummer (vier sedezimale Ziffern)
Bytes 6 und 7:	Maximale Länge der erweiterten Maschinenanzeigeleiste [1] als ganze Dualzahl im Halbwortformat

Die Modellnummer für ein Modell 145 der IBM /370 ist z.B. 0145; für die übrigen Modelle gilt dasselbe Prinzip. Die Identifikationsnummer ist gleich der Serien- oder Fabriknummer der betreffenden Zentraleinheit; so wird z.B. die Fabriknummer 10102 zu 010102 gespeichert. Unter der Versionsnummer versteht man einen Schlüssel, der über den technischen Ausrüstungsstand des Computers Auskunft gibt. Man vergegenwärtige sich hierzu, daß an Computern stets dann Änderungen vorgenommen werden, wenn dies zur Verbesserung ihrer Leistungsfähigkeit oder zur Anhebung ihrer Zuverlässigkeit und Betriebssicherheit für zweckmäßig erachtet wird, selbst wenn sie bereits installiert sind. Der Grund für solche Änderungen kann im simpelsten Falle auch darin liegen, daß dem Wartungspersonal mehr und bessere Informationen über den Zustand der Hardware zugänglich gemacht werden sollen.

Die erweiterte Maschinenanzeigeleiste nimmt nützliche Informationen für das Wartungspersonal auf. Sie werden auf den Hauptspeicherplätzen gespeichert, die beim Platz mit der im Steuerregister 15 enthaltenen Adresse beginnen, und zwar immer dann, wenn ein Maschinenfehler aufgetreten ist. Die Anzahl der auf diese Weise in den Hauptspeicher hinausgeschobenen Bytes hängt von der Art des Maschinenfehlers, von der Modellnummer und sogar von der Versionsnummer ab, d.h. vom technischen Ausrüstungsstand der Maschine. Obwohl diese Leiste eigentlich überall abgespeichert werden kann, wird sie in der Regel auf den bei der Adresse 000200 beginnenden Hauptspeicherplätzen. abgelegt.

Diese Instruktion sowie die Instruktion „Speichern Identifizierung des Kanals (Store Channel Identification)" mit dem mnemonischen Operationsschlüssel STIDC ermöglichen den Ablauf der gleichen Diagnostikprogramme auf allen Modellen der Computerfamilie IBM /370. Diese zwei Instruktionen befähigen die Programme, genügend Speicherplatz für die Behandlung von Hardwareproblemen zu reservieren. Darüberhinaus ist in ihnen Vorsorge getroffen worden, daß modellabhängige Routinen von einem Plattenspeicher in den Hauptspeicher geladen werden, die die in der Anzeigeleiste stehenden Daten weiterverarbeiten und analysieren.

---

1) ZE ist die Abkürzung von Zentraleinheit.
CPU ist die Abkürzung von Central Processing Unit.

2) Im angloamerikanischen Sprachraum als Machine Check Extended Logout (abgekürzt MCEL) bezeichnet.

Aufgabengruppe 19

1) Gegeben sind Inhalte der folgenden Register:

Steuerregister	0	ØØ4Ø ØØEØ
Steuerregister	1	Ø1ØØ 8ØØØ
Allgemeines Register	1	ØØØØ CE42
Allgemeines Register	2	8ØØØ 4ØØØ
Allgemeines Register	3	ØØØØ EØØØ

Ferner sind die Inhalte der folgenden Hauptspeicherplätze bekannt:

Hauptspeicherplätze				Inhalte							
ØØ8 ØØØ	bis	ØØ8 ØØF	FØØØ	8Ø8Ø	FØØØ	8ØCØ	FØØØ	81ØØ	FØØØ	814Ø
ØØ8 Ø1Ø	bis	ØØ8 Ø1F	FØØØ	818Ø	FØØØ	81CØ	FØØØ	82ØØ	FØØØ	824Ø
ØØ8 Ø2Ø	bis	ØØ8 Ø2F	FØØØ	828Ø	FØØØ	82CØ	FØØØ	83ØØ	FØØØ	834Ø
ØØ8 Ø3Ø	bis	ØØ8 Ø3F	FØØØ	838Ø	FØØØ	83CØ	FØØØ	84ØØ	FØØØ	844Ø
ØØ8 Ø4Ø	bis	ØØ8 Ø4F	FØØØ	848Ø	FØØØ	84CØ	FØØØ	854Ø	FØØØ	85ØØ
ØØ8 Ø5Ø	bis	ØØ8 Ø5F	FØØØ	8581	FØØØ	85C1	FØØØ	8641	FØØØ	86Ø1
ØØ8 Ø6Ø	bis	ØØ8 Ø6F	FØØØ	8681	FØØØ	86C1	FØØØ	86Ø1	FØØØ	8641
ØØ8 Ø7Ø	bis	ØØ8 Ø7F	FØØØ	87Ø1	FØØØ	87Ø1	FØØØ	87Ø1	FØØØ	87Ø1
ØØ8 Ø8Ø	bis	ØØ8 Ø8F	ØØ9C	ØØEØ	ØØE8	ØØFØ	ØØF8	Ø1ØØ	Ø13Ø	Ø138
ØØ8 Ø9Ø	bis	ØØ8 Ø9F	ØØ9Ø	Ø1Ø8	Ø11Ø	Ø118	Ø12Ø	Ø128	Ø134	Ø13C
ØØ8 ØAØ	bis	ØØ8 ØAF	Ø14Ø	Ø144	ØØØ4	ØØØC	ØØØE	ØØØ4	ØØØC	Ø148
ØØ8 ØBØ	bis	ØØ8 ØBF	Ø15Ø	ØØA2	Ø154	Ø158	Ø15C	Ø16Ø	Ø168	Ø16C
ØØ8 ØCØ	bis	ØØ8 ØCF	Ø17Ø	ØØB8	Ø178	ØØA4	ØØA8	ØØAC	Ø18Ø	Ø188
ØØ8 ØDØ	bis	ØØ8 ØDF	Ø194	Ø194	Ø194	Ø194	Ø194	Ø194	Ø194	Ø194
ØØ8 ØEØ	bis	ØØ8 ØEF	Ø19Ø	Ø198	Ø1AØ	Ø1A8	ØØB4	ØØB4	ØØB4	ØØB4
ØØ8 ØFØ	bis	ØØ8 ØFF	Ø1BØ	Ø1B8	Ø1CØ	Ø1C8	Ø1DØ	Ø1D8	Ø1EØ	Ø1E8

a) Wie groß ist die in Gebrauch befindliche Seitengröße?
b) Wie groß ist die in Gebrauch befindliche Segmentgröße?
c) Aus den in den Steuerregistern enthaltenen Informationen sind Rückschlüsse über den Umfang des Programms zu ziehen.

2) Bezugnehmend auf die Aufgabe 1) sind die realen Adressen zu ermitteln, die in die folgenden Instruktionen verwickelt sind, bzw. auf welche Unterbrechungen diese Instruktionen führen!

a) L ¦ 58 4Ø B ØØØ
b) L ¦ 58 41 2 ØØØ
c) L ¦ 58 41 3 ØØØ
d) L ¦ 58 4 1 Ø ØØØ
e) L ¦ 58 4 2 3 ØØØ
f) L ¦ 58 4 1 2 ØØØ

3) Bezugnehmend auf die Aufgabe 1) ist die Frage zu beantworten, welchen Inhalt das allgemeine Register 4 nach Ausführung der nachfolgenden Instruktionen aufweist!

a) LA ¦ 41 4 1 2 ØØ2
b) LRA ¦ B1 4 1 2 ØØØ
c) LRA ¦ B1 4 2 3 232

8.9. Zeitgeber

Alle Modelle der IBM /370 sind standardmäßig mit einem Intervallzeitgeber und einem Uhrzeitgeber ausgerüstet.

Das Wort mit der sedezimalen Adresse 000050 (80 dezimal) enthält auf seinen ersten 24 Bitstellen den laufenden Wert des Intervallzeitgebers. Jede 1/300 Sekunde holt sich der Intervallzeitgeber dieses Wort, subtrahiert von ihm eine 1 in der Bitstelle 23 und speichert es anschließend zurück. Wenn dabei der Zeitwert negativ wird, so kommt es zu einer externen Unterbrechung. Diese kann unterdrückt werden, wenn der Bitstelle 24 des Steuerregisters 0 eine 1 zugewiesen ist. Selbstverständlich können nach wie vor alle externen Unterbrechungen gemeinsam durch Zuweisung einer 1 zur Bitstelle 7 des Programmstatuswortes (PSW) gesperrt werden, das zur Erinnerung.

Bei einigen Modellen wird der Zeitgeber häufiger, bei anderen weniger oft angesprochen, der Effekt ist jedoch stets der gleiche: Von der Bitstelle 23 des Wortes mit der sedezimalen Adresse 000050 wird praktisch 300 mal in der Sekunde 1 subtrahiert. Der Zeitgeber wird nur zwischen zwei Instruktionen oder während der Ausführung der beiden Instruktionen MVCL und CLCL vermindert. Sollte eine Kanalaktivität den Hauptspeicher für eine längere Zeit als $3\frac{1}{3}$ ms vollständig beanspruchen, so ist der Intervallzeitgeber nicht in der Lage, zum Hauptspeicherplatz mit der sed. Adresse 000050 zuzugreifen. Infolgedessen kann der dort stehende Zeitwert auch nicht vermindert werden, was wiederum einen Zeitgenauigkeitsverlust zur Folge hat. Der Zeitgeber ruht ebenfalls, wenn sich der Computer im gestoppten Zustand befindet.

Der Uhrzeitgeber belegt ein internes Doppelwort im Computer. Jede Mikrosekunde wird auf die Bitstelle 51 dieses Doppelwortes eine 1 addiert. Bei einigen Modellen wird der Uhrzeitgeber häufiger erhöht, doch bleibt auch bei ihnen die Wirkung die gleiche: Jede Mikrosekunde erfolgt die Addition einer 1 auf die Bitstelle 51. Rechnen wir einmal um: Die soeben geschilderte Erhöhung führt praktisch dazu, daß zur Bitstelle 31 des Doppelwortes nach jeweils 1,048576 Sekunden eine 1 addiert wird. Bei einigen Modellen wird dem Uhrzeitgeber die Spannung entzogen, wenn der Computer abgeschaltet wird, bei anderen Modellen hingegen läuft der Uhrzeitgeber weiter. Beim Einschalten werden alle Bitstellen des Uhrzeitgebers auf 0 gesetzt. Man ist übereingekommen, daß dieser Nullwert folgendem Zeitpunkt entspricht: 1. Januar 1900, 0.00 Uhr. Schließt man sich diesem Übereinkommen an, so wird der Bitstelle 0 des Uhrzeitgebers gerade am 11. Mai 1971 eine 1 zugewiesen, d.h. die Gesamtlaufzeit des Uhrzeitgebers ist nahezu 143 Jahre. Der Uhrzeitgeberwert wird, das sollte eigentlich schon vorausgeschickt sein, als ganze Dualzahl ohne Vorzeichen betrachtet.

Zwei Instruktionen ermöglichen den Zugriff zum Uhrzeitgeber. Sie lauten:

a)	Setzen Uhrzeitgeber —— Set Clock	SCK	B204	S
b)	Speichern Uhrzeitgeber —— Store Clock	STCK	B205	S

Die Instruktion SCK ist eine privilegierte Instruktion, die Instruktion STCK hingegen nicht. Der durch die effektiven Adressen beider Instruktionen angesprochene Operand wird als Doppelwort aufgefaßt. Da SCK eine privilegierte Instruktion ist, muß bei ihr der Operand auf Doppelwortgrenze ausgerichtet sein. Bei der Instruktion STCK ist diese Bedingung hinfällig, da bei ihr die Einrichtung zum Tragen kommt, daß alle Operanden byteorientiert sein können, d.h. auf einem beliebigen Hauptspeicherplatz beginnen können.

Mittels der Instruktion SCK kann dem Uhrzeitgeber ein neuer Wert zugewiesen werden. Befindet sich der Uhrzeitgeberschalter (UHRZEITG) am Bedienerkonsol während der Ausführung dieser Instruktion in der Stellung SETZEN WIRKSAM, so wird der Uhrzeitgeber auf den neuen Wert und der Bedingungsschlüssel auf 0 gesetzt. Liegt dieser

Schalter jedoch in seiner Normalstellung (GESICHERT), so wird die Wertzuweisung unterdrückt und der Bedingungsschlüssel auf 1 gesetzt. Ist der Uhrzeitgeber nicht betriebsbereit, so wird der Wert 3 für den Bedingungsschlüssel zurückgegeben.

Wenn zum ersten Mal Spannung an den Uhrzeitgeber gelegt wird, so gilt dieser als nicht gesetzt. Erst die erste erfolgreiche Ausführung der SCK-Instruktion führt ihn in den gesetzten Zustand über. Sollte ein Maschinenfehler den Uhrzeitgeberwert zerstören, so nimmt der Uhrzeitgeber den Fehlerstatus an. Verursacht ein Defekt, daß der Uhrzeitgeber seine Funktion einstellt, so wird er in den nicht betriebsbereiten Zustand versetzt.

Die Ausführung der Instruktion STCK bewirkt, daß der Uhrzeitgeberwert auf den Hauptspeicherplatz gespeichert wird, dessen Adresse aus den Bestandteilen des Operanden gebildet wird. Voraussetzung hierfür ist, daß sich der Uhrzeitgeber im gesetzten Zustand befindet, was sich auch im Bedingungsschlüssel 0 ausdrückt. Im nichtgesetzten Zustand hingegen wird der Bedingungsschlüssel 1 zurückgeliefert. Im Fehlerstatus wird der Bedingungsschlüssel auf 2 gesetzt; außerdem ist der abgespeicherte Wert nicht zuverlässig, d.h. nicht brauchbar. Befindet sich schließlich der Uhrzeitgeber im nicht betriebsbereiten Zustand, so wird der Bedingungsschlüssel auf 3 gesetzt und lauter Nullen den Bitstellen zugewiesen, die die durch den Operanden spezifizierten Hauptspeicherplätze ausmachen.

8.10. Zusätzliche Zeitmeßeinrichtungen

Ein Uhrzeitvergleicher ist bei den meisten Modellen eine Zusatzeinrichtung, die gesondert bestellt werden muß. Einige Modelle sind jedoch standardmäßig mit ihm ausgerüstet, bei einigen Modellen ist er nicht verfügbar. Er wird hauptsächlich bei größeren Modellen benutzt.

Der Uhrzeitvergleicher ist durch ein internes Doppelwort realisiert, das einen Wert in der gleichen Form wie beim Uhrzeitgeber aufnimmt. Zwei privilegierte Instruktionen sprechen den Uhrzeitvergleicher an. Sie lauten:

a)	Setzen Uhrzeitvergleicher —— Set Clock Comparator	SCKC	B206	S
b)	Speichern Uhrzeitvergleicher —— Store Clock Comparator	STCKC	B207	S

SCKC stellt das adressierte Doppelwort in den Uhrzeitvergleicher. Die Bitstellen, die den beim Uhrzeitgeber nicht benutzten rechtsbündigen Bitstellen entsprechen, werden dabei auf Null gesetzt. STCKC speichert den gegenwärtigen Wert des Uhrzeitvergleichers in das durch die Instruktion angesprochene Doppelwort. Die rechtsbündigen Bitstellen dieses Doppelwortes, die den beim Uhrzeitgeber nicht benutzten entsprechen, werden hierbei auf Null gesetzt.

In folgenden Fällen geschieht eine externe Unterbrechung, es sei denn, ihr Auftreten wird durch eine 1 auf der Bitstelle 20 des Steuerregisters 0 verhindert (maskiert):

a) Der Uhrzeitgeber befindet sich im Fehlerstatus.
b) Der Uhrzeitgeber befindet sich im nicht betriebsbereiten Status.
c) Der Uhrzeitgeber befindet sich im gesetzten oder nicht gesetzten Zustand und der Wert des Uhrzeitvergleichers ist kleiner als der Wert des Uhrzeitgebers. Beide Werte gehen dabei als vorzeichenlose ganzzahlige Dualzahlen in den Vergleich ein. Im Grunde genommen bedeutet das, daß man in den Uhrzeitvergleicher die Zeit hineinbringt, zu der eine Unterbrechung gewünscht wird.

Es ist stets zu beachten, daß die Bedingung, die eine Unterbrechung verursacht hat, nur durch das Zurücksetzen des Uhrzeitgebers oder durch die Zuweisung eines neuen höheren Wertes zum Uhrzeitvergleicher beseitigt werden kann. Geschieht das nicht, so werden fortlaufend Unterbrechungen generiert, es sei denn, diese sind maskiert.

Ein ZE-Zeitgeber [1] ist bei den meisten Modellen als Zusatzeinrichtung verfügbar, bei einigen gehört er sogar zur Standardausrüstung. In einige wenige Modelle kann er nicht eingebaut werden. Seine Verwendung erfolgt vorzugsweise bei den größeren Modellen.

Der ZE-Zeitgeber ist durch ein internes Doppelwort verwirklicht, so wie es beim Uhrzeitgeber geschehen ist. Der Wert des ZE-Zeitgebers wird zu den Zeitpunkten vermindert, in denen der Wert des Uhrzeitgebers erhöht wird. Die Verminderung kann in Perioden hoher Kanalaktivität unterbleiben; im gestoppten Zustand des Computers wird sie auf jeden Fall unterdrückt. Immer dann, wenn auf der Bitstelle 0 des Zeitgebers eine 1 entsteht (als Folge eines „Borgens"), kommt es zum Auftreten einer Unterbrechung, es sei denn, diese ist durch die Bitstelle 21 des Steuerregisters 0 maskiert. Wird die Unterbrechung verhindert, so bleibt sie nicht hängen; es werden jedoch solange Unterbrechungsbedingungen erzeugt, bis die Bitstelle 0 wieder auf 0 zu stehen kommt. Zwei Instruktionen sprechen den ZE-Zeitgeber an:

a)	Setzen ZE-Zeitgeber —— Set CPU Timer	SPT	B208	S
b)	Speichern ZE-Zeitgeber —— Store CPU Timer	STPT	B209	S

Durch SPT wird der Inhalt des ZE-Zeitgebers durch den Inhalt des Doppelwortes ersetzt, das durch den Operanden spezifiziert ist. Dabei werden die rechtsbündigen Bitstellen des ZE-Zeitgebers, die den nicht benutzten des Uhrzeitgebers entsprechen, auf 0 gesetzt. Durch die Instruktion STPT wird der Wert des ZE-Zeitgebers in das durch den Operanden angesprochene Doppelwort gespeichert. Die rechtsbündigen Bitstellen dieses Doppelwortes, die den nicht benutzten des Uhrzeitgebers entsprechen, werden dabei auf Null gesetzt. Beide Instruktionen sind privilegierte Instruktionen.

Der ZE-Zeitgeber führt im Grunde genommen die gleichen Funktionen aus wie der Intervallzeitgeber, doch weist er eine größere Genauigkeit auf.

8.11. Mehrprozessorsysteme[2]

Ein Mehrprozessorsystem besteht aus einem Verbund von zwei oder mehr Zentraleinheiten, die gemeinsam den gleichen Hauptspeicher benutzen. Eine gründliche Diskussion solcher Systeme würde den Rahmen dieses Buches sprengen. Deshalb beschränken wir uns auf die Erwähnung ihrer Charakteristiken.

Mehrprozessorsysteme sind hauptsächlich durch folgende Eigenschaften charakterisiert:

1) Jede Zentraleinheit (ZE) kann zum gesamten Hauptspeicher zugreifen. Mehrprozessorsysteme auf der Basis der IBM /370 können mit Hauptspeichern zwischen 2 und 16 MB [3] ausgerüstet werden.

---

1) ZE ist hier die Abkürzung von Zentraleinheit, im angloamerikanischen Sprachraum CPU (Central Processing Unit).

2) Nach Definition ein digitales Rechensystem, bei dem ein Hauptspeicher ganz oder teilweise von zwei oder mehr Prozessoren gemeinsam benutzt wird, deren jeder über mindestens ein Rechenwerk und mindestens ein Leitwerk allein verfügt.

3) 1 MB ≙ 1 048 576 Bytes.

2) Entweder bedient eine Uhrzeitgeber alle Zentraleinheiten oder es sind zusätzliche Schaltkreise vorhanden, die für den Gleichlauf aller Uhrzeitgeber sorgen, so daß sie sämtlich auf die identische Zeit verweisen.

3) Jeder ZE ist ein Vorschaltregister hinzugefügt.

4) Jede ZE kennt vier zusätzliche Instruktionen.

Bei Einschaltung einer ZE wird das Vorschaltregister auf Null gesetzt. Die beiden Instruktionen

a)	Setzen Vorschaltregister —— Set Prefix	SPX	B210	S
b)	Speichern Vorschaltregister —— Store Prefix	STPX	B211	S

betreffen das Vorschaltregister. Die Instruktion SPX holt einen neuen Wert für das Vorschaltregister von den Bitstellen 8 bis 19 des vom Operanden angesprochenen Wortes. Umgekehrt speichert STPX den Inhalt des Vorschaltregisters auf die Bitstellen 8 bis 19 des adressierten Wortes, wobei dessen restlichen Bitstellen Nullen zugewiesen werden.

Liegt eine reale Adresse im Bereich von 0 bis 4095, so werden die Bits 8 bis 19 dieser Adresse durch die Bits 8 bis 19 des Vorschaltregisters ersetzt. Stimmen dagegen die Bits 8 bis 19 der realen Adresse mit den an den gleichen Stellen stehenden Bits des Vorschaltregisters überein, so werden sie durch Nullen ersetzt. Man bezeichnet eine reale Adresse, die dem Einfluß des Vorschaltregisters unterlegen hat, als absolute Adresse. Die Berücksichtigung des Vorschaltregisters erfolgt bei allen Operationen der Zentraleinheit, nicht jedoch bei Kanaloperationen (Adressen in Kanalbefehlswörtern, Adressen in indirekten Adreßwörtern usw.); die Adressen in Kanaladreßwörtern und in Kanalstatuswörtern werden jedoch nicht in diesem Sinne betrachtet. Da die Hauptspeicherplätze mit den Adressen 0 bis 4095 die Stellen mit fester Bedeutung (feste Stellen) einschließen, ergibt sich ein Problem, wenn eine ZE die festen Stellen der anderen Zentraleinheiten benutzen muß. Die Lösung besteht darin, daß jede ZE einen anderen Vorschaltwert verwendet. Für eine bestimmte ZE erscheinen dann die Adressen normal, die anderen ZE können auf diese Plätze mit Hilfe von absoluten Adressen zugreifen, da sie jeweils einen unterschiedlichen Vorschaltwert aufweisen.

Die beiden anderen Instruktionen heißen:

a)	Speichern der ZE-Adresse —— Store CPU Address	STAP	B212	S
b)	Signalisieren eines Prozessors —— Signal Processor	SIGP	AE	RS

Jeder ZE in einem Mehrprozessorsystem ist eine ZE-Adresse zugeteilt. Durch STAP wird diese Adresse in ein Halbwort gespeichert, das auf der Adresse liegt, die aus dem Operanden zusammengestellt wird. SIGP schickt ein Signal zu der ZE aus, deren Adresse in der rechten Hälfte des Registers R_3 gefunden wird. Das Signal ist ein Byte lang und wird von dem Speicherplatz her genommen, dessen Adresse aus dem Inhalt des Basisregisters B_2 und der Distanzadresse D_2 gebildet wird. Von der ZE, die das Signal empfängt, läuft ein Statuswort zurück, das im Register R_1 erscheint. Die ausgesendeten Signale können an die Stelle normaler Bedienereingriffe treten, wie z.B. an die Stelle des Drückens der Start- oder der Stoptaste. Sie können aber ebenfalls spezielle externe Unterbrechungen verursachen.

8.12. Die Handhabung der Speicherschutzschlüssel im Programmstatuswort

Der im Programmstatuswort (PSW) vorhandene Speicherschutzschlüssel kann durch zwei Instruktionen angesprochen werden, mit denen alle Mehrprozessorsysteme standardmäßig ausgerüstet sind, bei den Einprozessorsystemen sind sie als Zusatzeinrichtung verfügbar. Wie die Mehrprozessorinstruktionen sind auch diese beiden Instruktionen privilegiert.

Die beiden Instruktionen heißen:

a)	Setzen PSW-Schlüssel von der Adresse — Set PSW Key from Address	SPKA	B20A	S

b)	Einfügen PSW-Schlüssel — Insert PSW Key	IPK	B20B	S

Bei der Instruktion IPK wird die Operandenadresse nicht benutzt. Durch IPK wird der vier Bits umfassende Speicherschutzschlüssel, der im laufenden PSW enthalten ist, auf die Bitstellen 24 bis 27 des allgemeinen Registers 2 gestellt. Die übrigen Bitstellen dieses Registers werden wie folgt behandelt: Die Bitstellen 24 bis 27 werden auf 0 gesetzt, die Bitstellen 0 bis 23 werden nicht verändert.

Durch die Instruktion SPKA wird der Speicherschutzschlüssel des laufenden PSW durch den Inhalt der Bitstellen 24 bis 27 der gebildeten Operandenadresse ersetzt, d.h. aus dem aus dem Inhalt des Basisregisters B_2 und der Distanzadresse D_2 zusammengefügten Wert. Es ist also zu beachten, daß durch den Operanden keine Daten adressiert werden, sondern daß er unmittelbar selbst den Speicherschutzschlüssel enthält; es liegt somit ein Instruktionsaufbau vor, der dem der EA- bzw. Schiftinstruktionen gleichkommt. In der Regel wird die Instruktion SPKA bezüglich B_2 und D_2 mit 2000 (sed) codiert werden. Damit ist es möglich, sich auf den gleichen Schlüssel zu beziehen, der zuvor durch die IPK-Instruktion ins allgemeine Register 2 abgelegt wurde. Vergegenwärtigen wir uns, daß ja 2000 (sed) besagt, daß die Distanzadresse D_2 auf den Inhalt des Basisregisters B_2 (hier also: 2) addiert wird.

Abgesehen von Laden eines neuen PSW sind somit diese beiden Instruktionen die einzigen, mit deren Hilfe der Speicherschutzschlüssel im PSW inspiziert oder geändert werden kann.

8.13. Instruktionen zur Synchronisierung

Drei nichtprivilegierte Instruktionen besitzen bei Mehrprozessorsystemen eine gewisse Bedeutung. Sie lauten:

a)	Testen und Setzen — Test and Set	TS	93	S

b)	Vergleichen und Vertauschen — Compare and Swap	CS	BA	RS

c)	Vergleichen doppelt und vertauschen — Compare Double and Swap	CDS	BB	RS

Bei allen Modellen der IBM /370 ist die TS-Instruktion standardmäßig vorhanden. Hinsichtlich der beiden übrigen Instruktionen CS und CDS ist zu sagen, daß sie bei einigen Modellen fest eingebaut sind, bei einigen Modellen sind sie als Sondereinrichtung

verfügbar, und einige können mit diesen Instruktionen überhaupt nicht ausgerüstet werden.

Bei der TS-Instruktion bleibt das Byte 1 unbeachtet, d.h. das auf den Operationsschlüssel folgende Byte. Das durch die effektive Adresse angesprochene Byte wird benutzt, um den Bedingungsschlüssel zu setzen; anschließend wird es durch ein Byte ersetzt, das auf allen Bitstellen eine 1 aufweist. Der Bedingungsschlüssel wird auf 0 oder 1 gesetzt, je nachdem, ob die am weitesten links stehende Bitposition des untersuchten Bytes eine 0 oder eine 1 enthält.

Bei Ausführung der Instruktion CS wird der erste Operand mit dem zweiten Operanden verglichen. Führt der Vergleich auf die Aussage „gleich", so ersetzt der dritte Operand den zweiten Operanden. Ist das Vergleichsergebnis dagegen „ungleich", so wird der zweite Operand auf den Platz des ersten gebracht. Symbolisch stellt sich der Vorgang wie folgt dar:

I. $(R_1) - (E_2)$
II. $= \Rightarrow (R3) \longrightarrow (E_2)$
III. $\neg = \Rightarrow (E_2) \longrightarrow (R_1)$

Bei Gleichheit der Operanden wird der Bedingungsschlüssel auf 0 gesetzt, bei Ungleichheit auf 1. Der zweite Operand muß ein Wort umfassen und an einer Wortgrenze liegen.

Die Instruktion CDS funktioniert im Prinzip wie die Instruktion CS. Die von ihr betroffenen Operanden müssen jedoch folgenden Voraussetzungen genügen: Der zweite Operand muß gleich einem Doppelwort sein und an einer Doppelwortgrenze untergebracht sein; die beiden übrigen Operanden (R_1 bzw. R_3) müssen als allgemeines Register mit gerader Ordnungsnummer ausgewiesen sein, d.h. durch sie wird ein Registerpaar angesprochen (geradzahlig-ungeradzahlig).

Als wesentliches Merkmal dieser drei Instruktionen sowie der nichtprivilegierten Instruktion STCK und der privilegierten Instruktionen LPSW, RDD, WRD, SPX und SIGP ist ferner anzusehen, daß sie eine zeitliche Aufeinanderfolge von Operationen erzwingen; das geschieht während ihrer Ausführung. Wenn eine der soeben genannten Instruktionen abläuft, so werden zunächst einmal alle noch ausstehenden Hauptspeicherzugriffe befriedigt. Erst danach wird die betreffende Instruktion ausgeführt. Zusätzlich wird die Entschlüsselung nachfolgender Instruktionen solange verhindert, bis die durch die fraglichen Instruktionen initiierten Operationen vollständig abgelaufen sind. Bei einem Mehrprozessorsystem hat das zur Folge, daß bei allen gekoppelten Zentraleinheiten die Ausführung von Instruktionen solange aufhört, bis die mit einer solchen Instruktion beschäftigte Zentraleinheit diese abgeschlossen hat. Kanäle werden von dieser Erzwingung des zeitlichen Nacheinanders nicht betroffen.

Da diese Instruktionen scheinbar alles zum Stillstand bringen, benötigen sie auch eine entsprechend höhere Ausführungszeit. Jedoch stellen sie ein probates Mittel dar, durch das eine Zentraleinheit Informationen in ein gemeinsam benutztes Hauptspeichergebiet hineinstellen kann, von dem aus sie andere Zentraleinheiten abrufen können. Da Kanaloperationen nicht angehalten werden, sollte dieses gemeinsam benutzte Hauptspeichergebiet nicht für EA-Zwecke eingesetzt werden.

8.14. Der Monitoraufruf

Die Instruktion

Monitoraufruf —— Monitor Call	MC	AF	SI

ist eine nichtprivilegierte Instruktion. Sie verursacht bei ihrer Ausführung das Auftreten einer Spezifikations-Ausnahmebedingung, wenn der Wert des Direktbytes der Instruktion 15 übersteigt. Die durch diese Instruktion hervorgerufene Operation berührt die rechte Hälfte des Steuerregisters 8, d.h. dessen Bitstellen 16 bis 31.

Das Byte I_2 in dieser Instruktion muß auf einen Wert zwischen 0 und 15 verweisen. Die Bitstellen 16 bis 31 des Steuerregisters 8 entsprechen den Werten 0 bis 15 des Direktbytes I_2. Wenn die einem bestimmten Wert von I_2 entsprechende Bitstelle des Steuerregisters 8 eine 1 aufweist, so kommt es zum Auftreten einer Programmunterbrechung. Dabei wird die in üblicher Weise aus B_1 und D_1 gebildete Operandenadresse auf die Hauptspeicherplätze mit den sedezimalen Adressen 00009C bis 00009F weggelegt, wobei dem Byte mit der niedrigsten Sedezimaladresse 00009C lauter Nullen zugewiesen werden. Der auf diese Weise weggespeicherte Wert heißt Monitorschlüssel.

Aufgabe dieser Instruktion ist es, statistische Daten über bestimmte Programmteile zu liefern. Wird das Steuerregister 8 auf Null gesetzt, so kommt keine Operation zustande. Durch Zuweisung einer 1 zur geeigneten Bitstelle des Steuerregisters 8 wird erreicht, daß eine bestimmte Information während der Programmausführung zur Geltung kommt. Wird die betreffende Information nicht gewünscht, so genügt es, den Inhalt des Steuerregisters 8 zu ändern, das Programm hingegen kann ungeändert beibehalten werden.

Zur Auffrischung unserer Kenntnisse wollen wir den Aufbau der Instruktionen des SI-Formates in Abb. 8.9. noch einmal wiederholen:

OP	I_2	B_1	D_1

Zeichenerklärungen:			
	OP	Operationsschlüssel	
	I_2	Direktoperand (Direktbyte)	
	B_1	Basisregister	des 1. Operanden
	D_1	Distanzadresse	des 1. Operanden

Abb. 8.9. Instruktionen im SI-Format

Die Monitoreinrichtung erlaubt also die Unterscheidung von 16 Monitorklassen (Wertevorrat 0 bis 15 im Byte I_2). Jedes Monitorereignis ist exakt erkennbar aus der Monitorklasse und dem Monitorschlüssel, der gleich der Summe aus dem Inhalt des Basisregisters B_1 und der Distanzadresse D_1 ist. Wenn der Inhalt von B_1 gleich Null ist, dann können 4096 Ereignisse pro Klasse definiert werden, bei 16 Klassen also 65536 Ereignisse. Ist dagegen der Inhalt von B_1 größer als 0, so können maximal $(2^{24}-1)\cdot 16$ Vorkommnisse festgestellt werden. Diese Zahlenspielereien mögen zur Charakterisierung der Mächtigkeit der Monitoraufruf-Instruktion beitragen.

8.15. Die Programmereignis-Registrierung

Die Programmereignis-Registrierung [1], abgekürzt PER, schließt keine neuen Instruktionen ein. Sie kann durch eine 1 in der Bitstelle 1 des Programmstatuswortes (PSW) beim erweiterten Steuerungsmodus verhindert (maskiert) werden. Unterbrechungen, die auf die PER zurückzuführen sind, sind im Basissteuerungsmodus nicht erlaubt. Durch den Inhalt der Bitstellen 0 bis 3 des Steuerregisters 9 wird festgelegt, welche Ereignistypen eine Unterbrechung verursachen sollen. Die Bitstellen 16 bis 31 des Steuerregisters 9 entsprechen den allgemeinen Registern 0 bis 15; eine 1 auf einer dieser Bitstellen hat zur Folge, daß für das mit ihr korrespondierende allgemeine Register eine PER-Unterbrechung erlaubt ist. In den Bitstellen 8 bis 31 des Steuerregisters 10 muß eine untere Hauptspeichergrenze, in den gleichen Bitstellen des Steuerregisters 11 eine obere Hauptspeichergrenze fixiert sein.

Durch eine 1 auf der Bitstelle 0 des Steuerregisters 9 wird festgelegt, daß erfolgreiche Verzweigungen aufgezeichnet werden sollen. Die Ausführung irgendeiner Verzweigungsinstruktion (BC, BCR, BAL, BALR, BCT, BCTR, BXH, BXLE) hat zur Folge, daß eine durchgeführte Verzweigung als Programmereignis registriert wird und in eine entsprechende Programmunterbrechung mündet.

Eine 1 auf der Bitstelle 1 des Steuerregisters 9 bewirkt, daß die Interpretation von bestimmten Instruktionen aufgezeichnet wird. Voraussetzung für eine solche Registrierung ist jedoch, daß das erste Byte einer solchen Instruktion auf einem Hauptspeicherplatz steht, dessen Adresse sowohl größer als oder gleich der im Steuerregister 10 enthaltenen Adresse als auch kleiner als oder gleich der im Steuerregister 11 vermerkten Adresse ist. Bei Erfüllung dieser Bedingungen kommt es zu einer PER-Unterbrechung. Bei der EX-Instruktion erfolgt die Unterbrechung, wenn entweder die EX-Instruktion selbst oder die von ihr beeinflußte Instruktion oder beide Instruktionen innerhalb des vorgegebenen Hauptspeichergebietes liegen.

Durch eine 1 auf der Bitstelle 2 des Steuerregisters 9 wird erreicht, daß Änderungen des Inhaltes bestimmter Hauptspeicherplätze registriert werden. Jede Einspeicherung in irgendeinen Hauptspeicherplatz innerhalb des durch die Inhalte der Steuerregister 10 und 11 angezeigten Hauptspeichergebietes gilt als Programmereignis und verursacht infolgedessen eine entsprechende Unterbrechung. Es zählen jedoch nur die Einspeicherungen, die von Instruktionen veranlaßt werden, nicht jedoch die, die auf Kanalbefehle zurückzuführen sind.

Eine 1 auf der Bitstelle 3 des Steuerregisters zeigt an, daß alle Änderungen des Inhaltes bestimmter allgemeiner Register aufgezeichnet werden sollen. Jede Ladeoperation, die auf ein allgemeines Register bezogen ist, dessen zugehörige Bitstelle im Steuerregister 9 eine 1 enthält, wird als Programmereignis betrachtet und führt infolgedessen zu einer Unterbrechung.

Änderungen des Inhalts von Hauptspeicherplätzen oder allgemeinen Registern werden immer dann vermerkt, wenn eine Instruktion Daten in das umrissene Hauptspeichergebiet oder die gekennzeichneten Register bewegt; dabei ist es ohne Belang, ob die betreffenden Daten zufällig mit dem vorhergehenden Inhalt übereinstimmen. Bei Ausführung der beiden Instruktionen MVCL und CLCL kommt es stets zur Registrierung eines neuen Ereignisses, wenn nach einer Unterbrechung die Ausführung dieser Instruktionen wieder aufgenommen wird.

---

1) Im angloamerikanischen Sprachraum als „Program Event Recording", abgekürzt PER bezeichnet.

Beim Auftreten einer PER-Unterbrechung wird der Sedezimalwert 0080 über die logische Oderoperation (OR) mit dem Programmunterbrechungsschlüssel verknüpft. Infolge der eingetretenen Unterbrechung werden zusätzlich die folgenden Hauptspeicherplätze mit Werten versorgt:

a) 000 096 und 000 097
Von den Bitstellen 0 bis 3 werden diejenigen auf 1 gesetzt, die den Ereignissen entsprechen, deren Auftreten die Unterbrechung verursachten. Dabei ist es wohl nach den vorhergehenden Erörterungen einleuchtend, daß als Voraussetzung für die Registrierung die ihnen entsprechenden Bitstellen des Steuerregisters 9 eine 1 enthalten müssen. Stets ist mindestens eine Bitstelle auf 1 gesetzt. Den Bitstellen 4 bis 15 wird generell eine Null zugewiesen.

b) 000 098 bis 000 09B
Die Hauptspeicherplätze mit diesen Adressen nehmen die Adresse der die Unterbrechung verursachenden Instruktion auf, wobei 000 098 vollständig auf 0 gesetzt wird. Bei einer EX-Instruktion wird die Adresse dieser Instruktion selbst gespeichert. Das PSW zeigt grundsätzlich auf die Adresse der nächsten auszuführenden Instruktion.

Alle Adressen, die unter a) und b) erwähnt sind, wurden in sedezimaler Form notiert.

Die Programmereignisregistrierung ist eine sicher sehr nützliche Hardwareeinrichtung, wenn es darum geht, Programme zu untersuchen, deren Ablauf erhebliche Sorgen bereitet. Wird die PER nicht benötigt, so kann sie durch eine 1 auf der Bitstelle 1 des PSW unterdrückt (maskiert) werden. Durch die Verwendung der PER-Einrichtung kann die Ablaufzeit eines Programmes anwachsen, sogar dann, wenn es zu keinen Unterbrechungen kommt.

8.16. Die Mikroprogrammierung

Die Ausführung einer Instruktion erfolgt in mehreren einzelnen Schritten. Bei jedem Schritt werden gewöhnlich die Daten von einem internen Register in ein anderes übertragen. Während eines solchen Transportes können verschiedene Operationen mit den Daten geschehen. So können z. B. ihre Komplementwerte gebildet werden oder sie können auf die Inhalte anderer interner Register addiert werden. Selbstverständlich ist es auch möglich, daß sie ungeändert bleiben. Der höchste Anteil an den Gesamtkosten für das Steuerwerk eines Computers fällt auf die Dioden und Transistoren, die zur Einhaltung der korrekten Datenpfade geschaltet werden müssen. Für jede Instruktion wird eine gesonderte Menge von Schaltkreisen benötigt, wenn auch einander sehr ähnliche Instruktionen eine gewisse Anzahl bestimmter Schaltkreise gemeinsam benutzen können.

Unter der Mikroprogrammierung wird eine Technik verstanden, mit deren Hilfe eine Steuerfunktion als eine Folge von in einem Speicher gespeicherten Steuersignalen entworfen und verwirklicht werden kann. Die elementarsten Funktionen wird man bei einem solchen Konzept wohl nach wie vor in Form von Schaltkreisen realisieren, aber die Herkunft der meisten Steuervorgänge ist in einem Speicher zu suchen. Die Ausführung einer bestimmten Instruktion besteht dann einfach in einer Verzweigung zum passenden Teil des Mikroprogrammspeichers.

In das Modell 195 der IBM /370 ist die Mikroprogrammtechnik nicht eingebaut. Die Forderung nach maximaler Geschwindigkeit bei nahezu keiner Rücksichtnahme auf

Kosten ließ die Konstrukteure die Verwirklichung aller Steuerfunktionen durch Schaltkreise als angemessen erscheinen. Das Modell 165 der IBM /370 besitzt einen Testspeicher [1] (unlöschbaren Speicher) für das Mikroprogramm. Hierunter versteht man einen speziellen Speicher, der im Herstellungswerk so eingebaut wird, daß es nur möglich ist, aus ihm Daten auszulesen, jedoch keine Daten in ihn hineinzuschreiben. Die meisten Modelle der IBM /370 sind jedoch mit Mikroprogrammen ausgerüstet, die im gleichen Speicherwerk untergebracht sind wie die Verarbeitungsprogramme. Beim Modell 145 der IBM /370 ist derjenige Teil des Hauptspeichers für Steuerfunktionen [2] reserviert, der im oberen Adreßbereich liegt.

Da viele Steuerspeicher ihres Inhaltes verlustig gehen, wenn der Computer ausgeschaltet wird, muß der Steuerspeicher bei Anlegen der Netzspannung wieder geladen werden. Eine kleine flexible Platte (ungefähr im Umfang einer mit 45 Umdrehungen/Minute laufenden Singleschallplatte) wird in Folge der Anlegung der Netzspannung automatisch gelesen. Der Inhalt dieser Platte wird in den Steuerspeicher geladen. Dieses einleitende Mikroprogrammladen [3] kann ebenso durch das Drücken einer speziellen Konsoltaste erreicht werden.

Nachdem die Idee mit der Platte einmal geboren war, war es eine natürliche Folgerung-Diagnostikplatten zu erzeugen, von denen aus spezielle Routinen in den Steuerspeicher geladen werden können. Einige solcher Routinen sind in der Lage, einen defekten Schaltkreis beinahe an jeder Stelle der Zentraleinheit zu lokalisieren. Besondere Einrichtungen können den Computern dadurch hinzugefügt werden, daß einfach eine neue Platte im Herstellungswerk erzeugt wird. In einigen Fällen muß dabei natürlich der Umfang des Steuerspeichers erhöht werden. Beim Modell 145 der IBM /370 z.B. verschiebt sich die Grenze zwischen dem Steuerspeicher und dem übrigen Teil des Hauptspeichers immer weiter nach unten, je mehr das Mikroprogramm anwächst. Damit ist ganz klar gesagt, daß der für die Verarbeitungsprogramme zur Verfügung stehende Hauptspeicherteil entsprechend kleiner wird.

Die Mikroprogrammierung kann heute als wirtschaftlichste Methode angesehen werden, mit deren Hilfe es möglich ist, auch kleinen und mittleren Computern eine mächtige Maschinensprache zu unterlegen. Sie hat sich inzwischen sogar bei einigen Steuereinheiten von Magnetplattenspeichergeräten bewährt.

8.17. Einleitendes Programmladen

Wenn der Computer eingeschaltet wird, müssen einige Vorkehrungen getroffen werden, um ein Programm in den Hauptspeicher zu laden und seine Ausführung einzuleiten. Das einleitende Programmladen [4] wird wie folgt durchgeführt: Mittels dreier am Konsol befindlichen Drehschalter wird eine Kanal-Geräte-Adresse eingestellt und die Ladetaste gedrückt. Die Drehschalter sind Teil des in Abb. 8.10. gezeigten Systemsteuerpultes.

Nun besitzen freilich einige Modelle der IBM /370 kein Systemsteuerpult im klassischen Sinne. Bei solchen Modellen muß zunächst die Taste BETRIEBSARTENAUSWAHL gedrückt werden. Danach ist der Buchstabe L einzugeben, wodurch eine IPL-Operation

1) Im angloamerikanischen Sprachraum als „Read only Storage", abgekürzt ROS, bezeichnet.
2) Der zur Aufnahme des Mikroprogrammes vorgesehene Teil des Hauptspeichers wird wegen der Steuerfunktionen des Mikroprogrammes oft als Steuerspeicher bezeichnet.
3) Im angloamerikanischen Sprachraum als „Initial Microprogram Load", abgekürzt IMPL, bezeichnet.
4) Im angloamerikanischen Sprachraum als „Initial Program Loading", abgekürzt IPL, bezeichnet.

angemeldet wird; in der Folge müssen die einzelnen Zeichen der Kanal-Geräte-Adresse eingetastet werden. Zum Schluß ist die Eingabetaste zu drücken.

Die IPL-Operation veranlaßt die Ausführung der folgenden Schritte:

1) Der Computer und alle an ihn angeschlossenen Geräte werden gelöscht. Alle im Gange befindlichen EA-Operationen werden abgestoppt. Der Bedingungsschlüssel und alle sonstigen Anzeiger werden gelöscht.

2) Der Computer liest von dem Eingabegerät, dessen Adresse entweder auf den drei Drehschaltern eingestellt oder eingetastet wurde, zwei Sätze, wobei der Befehlsschlüssel 02 benutzt wird. Der erste Satz wird dabei auf die Hauptspeicherplätze mit den sedezimalen Adressen 000 000 bis 000 017 einschließlich gelesen, d.h. immer 24 Bytes. Zum Lesen des 2. Satzes wird das Kanalbefehlswort (CCW) herangezogen, das auf dem Speicherplatz 000 008 beginnt. Gewöhnlich steht auf 000 010 ein weiteres Kanalbefehlswort, aber das muß nicht sein.

3) Nach Vollendung der Eingabeoperation wird das Programmstatuswort (PSW) vom Hauptspeicherplatz mit der Adresse 000 000 geladen und der Computer beginnt mit der Ausführung der Instruktionen.

8.18. Die Bedienung des Steuerpultes

Das in Abb. 8.9. schematisch dargestellte Systemsteuerpult zeigt die Schalter und die Anzeigelampen, die für die normale Bedienung der IBM /370 benötigt werden. Bei Modellen ohne Systemsteuerpult, z.B. beim Modell 115, werden dieselben Funktionen durch Drücken der Taste BETRIEBSARTENAUSWAHL mit nachfolgendem Eintasten geeigneter Antworten abgewickelt, wobei der Konsolbildschirm als visuelles Verbindungsglied wirkt. Die wichtigsten Tasten regeln die folgenden Funktionen (der besseren Orientierung wegen sind sowohl die angloamerikanische als auch die deutsch Beschriftung der Tasten aufgeführt):

1) POWER ON — NETZ EIN
Durch das Drücken dieser Taste erreichen wir, daß alle Anzeiger des Systems in Grundstellung gebracht und alle an das System angeschlossenen Geräte mit Spannung versorgt werden. Die Taste ist rückseitig beleuchtet, wenn das System am Stromnetz liegt. Durch das Niederdrücken dieser Taste werden keine Instruktionen ausgeführt; es wird jedoch das einleitende Mikroprogrammladen (IMPL) [1] verursacht.

2) POWER OFF — NETZ AUS
Durch das Drücken dieser Taste wird das System ausgeschaltet, d.h. vom Stromnetz getrennt. Die Inhalte einiger Hauptspeichertypen gehen beim Drücken dieser Taste verloren.

3) INTERRUPT — UNTERBRECHUNG
Das Niederdrücken dieser Taste bewirkt, daß eine externe Unterbrechung hervorgerufen wird. Diese kann durch eine 1 auf der Bitstelle 25 des Steuerregisters 0 unterdrückt (maskiert) werden. Der Unterbrechungsschlüssel ist 0040 (sed).

4) LOAD — LADEN
Durch das Drücken dieser Taste wird das einleitende Programmladen (IPL) [2] gestartet.

1) IMPL ist die Abkürzung von Initial Microprogram Loading.
2) IPL ist die Abkürzung von Initial Program Loading.

Auf dem Bedienersteuerpult sind weiterhin fünf Anzeigelampen untergebracht, die die folgenden Zustände signalisieren (auch hier ist neben der angloamerikanischen die deutsche Beschriftung aufgeführt):

1) SYSTEM — SYSTEM
   Diese Anzeigelampe leuchtet auf, wenn die Zeitmeßgeräte laufen, d.h. wenn entweder der Computer Instruktionen ausführt oder wenn irgendein EA-Gerät Daten überträgt.

2) MANUAL — MAN
   Diese Anzeigelampe leuchtet auf, wenn sich der Compter im gestoppten Zustand befindet. In diesen Zustand wird das System entweder durch Drücken der Stopptaste oder durch die erfolgreiche Ausführung des einleitenden Mikroprogrammladens oder durch eine mißglückte Ausführung des einleitenden Programmladens versetzt.

3) WAIT — WART
   Diese Lampe leuchtet auf, wenn sich der Computer im Wartestatus befindet. Das bedeutet, daß die Bitstelle 14 des PSW eine 1 enthält.

Wenn das System eingeschaltet ist, muß mindestens eine der drei eben besprochenen Anzeigenlampen aufleuchten.

4) TEST — TEST
   Zum Aufleuchten dieser Anzeigelampe kommt es, wenn sich irgendeine der manuellen Steuereinrichtungen des Systems nicht in ihrer Normalstellung befindet, oder wenn die Instruktion „Diagnose" abläuft, d.h. wenn eine Wartungsfunktion am System ausgeführt wird. Alle Hebelschalter befinden sich dann in Normalstellung, wenn sie horizontal (geradeaus) liegen, alle Drehschalter, wenn ihre Markierungen nach oben zeigen.

5) LOAD — LADEN
   Diese Lampe leuchtet auf, wenn die Ladetaste niedergedrückt wird; sie wird in dem Augenblick ausgeschaltet, in dem das Laden des neuen PSW erfolgreich abgeschlossen ist.

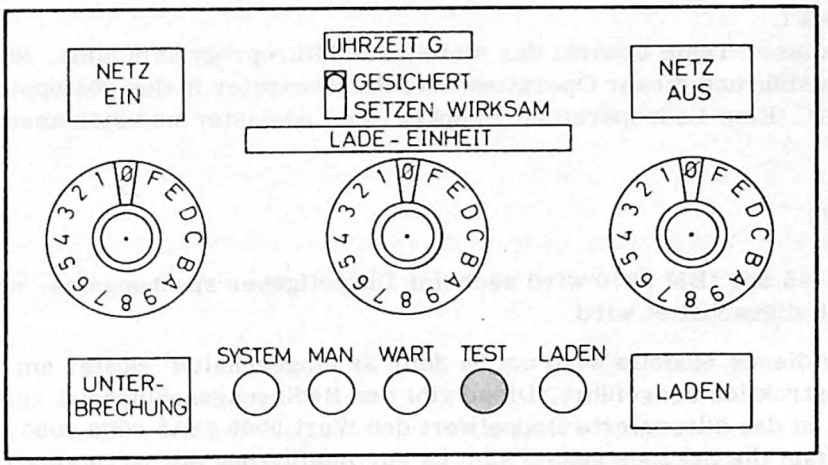


Abb. 8.10. Schematische Darstellung eines Systemsteuerpultes (Systemkonsole)

Neben diesen Schaltern und Anzeigelampen sind noch andere Steuereinrichtungen für das Bedienungs- und das Wartungspersonal in das Steuerpult eingebaut; der Umfang dieser Einrichtungen sowie ihre Anordnung ist bei den einzelnen Modellen unterschiedlich. Die wichtigsten Tasten und Schalter sowie ihre Funktion sind nachstehend

kurz beschrieben, wobei wiederum sowohl ihre angloamerikanische als auch ihre deutsche Beschriftung aufgeführt wird:

1) EMERGENCY — NOTHALT ZIEHEN
Durch Ziehen dieses Notschalters wird das System sofort vom Stromnetz getrennt. Nur das Wartunspersonal ist berechtigt und in der Lage, diesen Schalter wieder in seine Normallage zu bringen.

2) SYSTEM RESET — GRUNDSTG
Diese Drucktaste versetzt den Computer in den Stoppstatus, nachdem die ZE, die Kanäle ,ihre Steuereinheiten und EA-Geräte in ihre Grundstellung übergeführt wurden. Steuereinheiten und EA-Geräte, die von verschiedenen Computern gemeinsam benutzt werden, sind davon jedoch ausgenommen.

3) CPU RESET — ZE GRUNDSTG
Diese Drucktaste versetzt die ZE, jedoch nicht die Kanäle in die Grundstellung. Hängende Unterbrechungen und Fehleranzeiger werden ebenfalls zurückgesetzt.

4) CHECK RESET — FEHLER RÜCK
Durch diese Taste werden die Anzeiger für ZE- und Kanalfehler gelöscht.

5) STOP — STOP
Das Drücken dieser Taste bewirkt, daß die ZE in den gestoppten Zustand (Stoppstatus) versetzt wird. EA-Operationen werden abgeschlossen und hängende Unterbrechungen angenommen.

6) START — START
Das Drücken dieser nur im gestoppten Zustand funktionsfähigen Taste hat zur Folge, daß die Ausführung der Instruktionen fortgesetzt wird, gerade so, als ob die Stoptaste nicht gedrückt wurde.

7) RESTART — NEUSTART
Das Drücken dieser Taste führt dazu, daß einmal das laufende Programmstatuswort (PSW) auf den Speicherplatz 8 weggespeichert wird und zum anderen das PSW auf Speicherplatz 0 geladen wird.

8) IMPL — IMPL
Das Drücken dieser Taste bewirkt das einleitende Mikroprogrammladen. Nach erfolgreicher Ausführung dieser Operation wird der Computer in den gestoppten Zustand überführt. Eine Ladeoperation (Drücken der Ladetaste) muß sich anschließen.

Aufgabengruppe 20

1) Beim Modell 145 der IBM /370 wird auch der Uhrzeitgeber spannungslos, wenn die Zentraleinheit abgeschaltet wird.

Ein Computer dieses Modells wird um 8h 2min 3s eingeschaltet. Später am Tag wird eine STCK-Instruktion ausgeführt. Diese gibt den Bedingungsschlüssel 1 zurück und speichert in das adressierte Doppelwort den Wert 0000 38A5 0000 0000. Zu welcher Zeit (auf die Sekunde genau) kam es zur Ausführung der STCK-Instruktion?

2) Gegeben sind:
a) Laufendes PSW 473D 1FØØ ØØ23 23ØØ
b) die Inhalte der Register
Steuerregister 8 ØØØØ BØØØ
Steuerregister 9 1ØØØ B8ØØ

Allgem. Register 3 ØØ23 2ØØØ
Allgem. Register 4 ØØØØ 4ØØ8
Allgem. Register 5 ØØØØ 3ØØ4
Allgem. Register 6 Ø123 4567
Allgem. Register 7 89AB 4ØØ8

c) die Inhalte der Hauptspeicherplätze

232 ØØØ bis 232 ØØ7 Ø123 4567 89AB CDEF
232 ØØ8 bis 232 ØØF Ø123 4567 89AB 4ØØ8

Es sind der Wert des Bedingungsschlüssels sowie die Inhalte der geänderten Register und Speicherplätze nach Ausführung der folgenden Instruktionen anzugeben:

a) DCS BB 4 6 3 ØØ8
b) CD BA 6 4 3 ØØØ
c) TS 93 Ø Ø 3 ØØ6

3) Bezugnehmend auf Aufgabe 2) sind der Wert des Bedingungsschlüssels sowie die Inhalte der geänderten Register und Hauptspeicherplätze zu nennen, die nach Ausführung der folgenden Instruktionen zustande gekommen sind:

a) CDS BB 6 4 3 ØØ8
b) CD BA 4 6 3 ØØØ
c) CDS BB 4 6 3 ØØ4
d) L 58 Ø Ø 3 ØØØ
e) MC AF Ø Ø Ø Ø5Ø
f) MC AF Ø 1 5 Ø5Ø
g) LCTL B7 8 8 3 ØØ8

Außerdem sind die Werte anzugeben, die beim Auftreten einer Unterbrechung weggespeichert werden!

9. Grundlagen der Programmierung

9.1. Der Aufgabenkreis des Programmierers

Unter einem Programm versteht man eine Instruktionsmenge, die zur Lösung einer bestimmten Aufgabenstellung dient. Das Arbeitsverfahren, das zur Entwicklung und Herstellung eines solchen Programmes führt, bezeichnet man als Programmieren. Mit dem Programmieren und dem nachfolgenden Ablauf bzw. den nachfolgenden Abläufen des Programmes auf einem Computer ist es jedoch allein nicht getan. Ehe ein Programm auf einem Computer ausgeführt werden kann, müssen eine ganze Reihe von organisatorischen Vorkehrungen getroffen werden. Denken wir hierbei z. B. an ein Programm zur Lohnberechnung. Zunächst müssen die Zeitnachweise gesammelt werden; danach sind die in ihnen enthaltenen Informationen auf Lochkarten zu übertragen und schließlich müssen die Lochkarten auch geprüft und sortiert werden. Erst nach diesen vorbereitenden Arbeiten können diese Karten in das an einen Computer angeschlossene Kartenabfühlgerät eingelegt werden. Meist liegen zwischen ihnen noch Änderungskarten, die die persönlichen Verhältnisse der Mitarbeiter betreffen, wie z. B. Zahl der Familienangehörigen, Steuerfreibeträge usw. Nachdem der Computer die Überweisungsaufträge und die Kontrollabschnitte (Lohnbescheinigungen, Lohnzettel) ausgedruckt hat, müssen sich einige weitere organisatorische Arbeiten anschließen. Die Überweisungsaufträge müssen abgezeichnet, die einzelnen Aufträge von den anderen abgetrennt und schließlich verteilt werden. Bei jedem einzelnen Arbeitsschritt müssen überdies noch umfangreiche Sicherheitsvorkehrungen getroffen werden, damit einmal keine Aufträge verloren gehen und zum anderen, daß keine ungültigen oder fehlerhaften Belege entstehen und im Organisationsablauf verbleiben. Bei dem gesamten organisatorischen Prozeß, der bei der Lohnberechnung abläuft, ist also die Entwicklung des Computerprogrammes nur ein Problem unter vielen zu lösenden Problemen, freilich ein eminent wichtiges, kann doch ein solches Programm bei großen Firmen mit einer Vielzahl von in zahlreichen verschiedenen Berufen tätigen Arbeitnehmern durchaus die Größenordnung von 25 000 und mehr Instruktionen erreichen. Sowohl bei den kaufmännisch-kommerziellen als auch bei den technisch-wissenschaftlichen Anwendungen ist das Codieren, d.h. das Niederschreiben der Computerinstruktionen, nicht der schwierigste Teil des Programmierens. Viel schwieriger und vor allem auch zeitaufwendiger ist die Analyse der Aufgabenstellung, also das exakte Herausarbeiten der Antworten auf die alles entscheidenden Fragen, was eigentlich zu berechnen ist und wie es berechnet werden soll. Um diese Behauptung zu erhärten, wollen wir einen einfachen Fall betrachten. Ein umfangreiches Problem schließe die Bestimmung der Lösungen quadratischer Gleichungen ein. Es gibt mehrere Methoden, um solche Gleichungen mit Papier und Bleistift zu lösen, doch welche Methode ist für den Computer vorzuziehen? Was heißt eigentlich in diesem Zusammenhang "vorziehen", in welchem Sinn ist diese Vokabel gemeint? Soll das Programm mit möglichst wenig Speicherplatz auskommen, also statisch kurz sein, oder soll es möglichst rasch ablaufen, also dynamisch kurz sein, oder soll es gar so entworfen werden, daß es möglichst einfach codiert werden kann? Der Programmierer muß sich hier zu einer klaren, der Aufgabenstellung angepaßten Entscheidung durchringen.

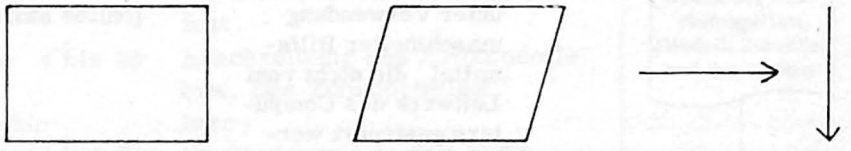
Der Programmierer braucht selbstverständlich seine Entscheidungen nicht auf einmal zu fällen. Er wird sich zunächst einen Überblick über das Gesamtsystem verschaffen müssen, einen Überblick über den gesamten organisatorischen Ablauf also. Der Computer wird hierbei nur als eine Einheit, als ein Baustein, unter vielen anderen organisatorischen Elementen betrachtet. Zweifellos fährt der Programmierer am besten, wenn er sich das Gesamtsystem zunächst einmal veranschaulicht, indem er einen sogenannten Datenflußplan zeichnet. Dieser Plan zeigt ein leicht verständliches Abbild des Gesamtgeschehens,

aufgegliedert in einzelne Elemente. Anschließend wird er sich jeden einzelnen Teil des Datenflußplanes gesondert vornehmen und wiederum in kleinere Schritte zerlegen; auf diese Weise erhält er detaillierte Datenflußpläne. Für den Teil der Datenverarbeitung, der für den Computer anfällt, kann er sich außerdem einen speziellen Programmablaufplan entwerfen und zeichnen. Dieser zeigt die einzelnen Schritte auf, mehr oder weniger gegliedert, in die das Computerprogramm zerfällt. Auch einen Programmablaufplan kann man im Bedarfsfall weiter verfeinern; von welcher Stufe der Zergliederung des Programmablaufplanes dann die Codierung beginnt, hängt sowohl von den Fähigkeiten des betreffenden Programmierers aber auch von einigen anderen Fakten ab, wie z. B. Schwierigkeit des Problems, Dokumentationserfordernisse u. a. mehr. Die Codierung kann von derselben Person ausgeführt werden, die das System analysiert hat, sie muß es aber nicht. Bei der Codierung wird einfach jeder Block des Programmablaufplanes in eine oder in mehrere Computerinstruktionen umgesetzt [1].

9.2. Datenflußpläne

Bei einem Datenflußplan liegt die Betonung auf den Belegen und Dokumenten, die einer Bearbeitung bzw. Verarbeitung unterzogen werden sollen sowie auf den Geräten und Einheiten, die zu den maschinellen Phasen herangezogen werden sollen. Auf die Angabe der Einzelheiten, d. h. wie die verschiedenen Operationen ausgeführt werden sollen, sollte sinnvollerweise jedoch verzichtet werden.

Zur graphischen Darstellung von Datenflußplänen und Programmablaufplänen werden drei Grundsymbole verwendet, die in Abb. 9.1. gezeigt werden. Durch ein Rechteck wird eine beliebige allgemeine Operation, ein Verarbeitungsschritt, dargestellt, durch ein Parallelogramm wird irgendeine Funktion eines EA-Gerätes gezeigt; durch die Pfeile wird die Reihenfolge der auszuführenden Operationen versinnbildlicht.



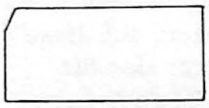
Allgemeine Operation (process) | Eingabe, Ausgabe (input/output) | Ablaufrichtung

Abb. 9.1. Grundsymbole für Datenflußpläne und Programmablaufpläne

Zusätzlich können gewisse andere Symbole zur Zeichnung von Datenflußplänen benutzt werden. Diese werden in Abb. 9.2. gezeigt. Die Sinnbilder sind gemäß DIN 66 001 vom September 1966 größtenteils genormt, wodurch der Austausch von Informationen zwischen den einzelnen Programmierern erheblich erleichtert wird.

---

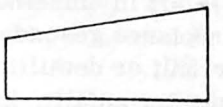
1) In der letzten Zeit hat sich meist eine Dreiteilung der Arbeit durchgesetzt. Der Systemanalytiker oder Organisator beschäftigt sich mit dem Entwurf des Gesamtsystems, dem Programmierer wird die Aufgliederung und Realisierung der Einzelschritte übertragen, während sich der Codierer mit der Niederschrift des Programmes befaßt.



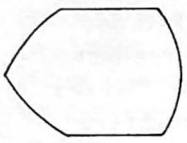
Lochkarte
(punched card)

Lochstreifen
(punched tape)

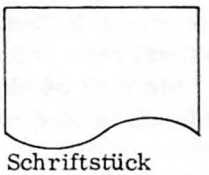
Datenübertragung
(communication link)



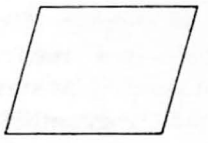
Eingeben von Hand
(manual input)



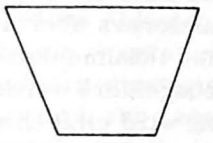
Anzeige (display)
in optischer oder
akustischer Form,
z.B. Ziffernanzeige,
Bildschirm, Kurven-
schreiber, Summer



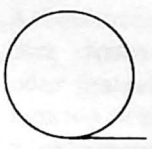
Schriftstück
(document)



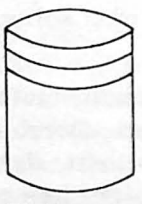
Datenträger,
allgemein
(input/output)



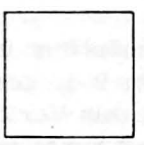
Eingreifen von Hand
(manual operation)
ohne Verwendung
maschineller Hilfs-
mittel, z.B. Ein-
tragungen in eine
Liste, Bandwechsel
usw.



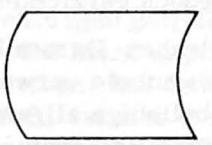
Magnetband
(magnetic tape)



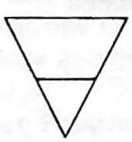
Plattenspeicher
(magnetic disk)



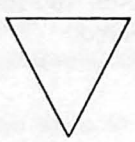
Ausführen einer
Hilfsfunktion
(auxiliary operation)
unter Verwendung
maschineller Hilfs-
mittel, die nicht vom
Leitwerk des Compu-
ters gesteuert wer-
den, z.B. das manuelle
Erstellen von Lochkar-
ten und Lochstreifen



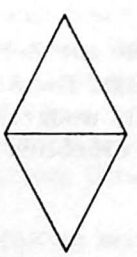
Datenträger, ge-
steuert vom Leit-
werk des Computers
(online storage)



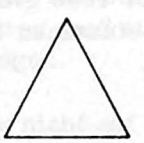
Datenträger, nicht
gesteuert vom Leit-
werk des Computers
(offline storage)
z.B. Ziehkartei



Mischen
(merge)



Sortieren
(sort)



Trennen
(extract)

Abb. 9.2. Sinnbilder für Datenflußpläne

An einem Beispiel wollen wir das Entwerfen von Datenflußplänen kennenlernen. Wir wollen dazu annehmen, daß der Inhaber der Bayerisch-Preußischen Autovertriebsgesellschaft in den USA sehr großen Ärger mit den Verkaufsabrechnungen seiner Verkäufer hat, die nicht selten ungenaue Preisnotierungen enthalten oder Fehler bei Summenbildungen; kein Wunder, denn oft sind Summen aus vielen Einzelposten zu bilden, da die Käufer beim Kauf eines Autos zahlreiche Zubehörteile mitkaufen. Der Inhaber hat sich deshalb entschieden, für die Verkaufsabrechungen eine IBM /370 einzusetzen. Er läßt deshalb Lochkarten ablochen, für jedes Automodell eine, für jedes Zubehörteil ebenfalls eine. Der Verkäufer braucht in Zukunft nur noch die passenden Karten aus einer Ziehkartei auszuwählen und in den Kartenleser des Computers einzulegen. Nachdem der Computer seinen Arbeitsgang erledigt hat, reißt er die vom Drucker ausgedruckte Seite einfach ab. Die Liste enthält zunächst eine Aufstellung aller gekauften Waren mit ihren Einzelpreisen, zum Schluß ist der Rechnungsbetrag gedruckt. Die Liste weist aber nicht nur die Verkaufspreise für die Wagen und die Zubehörteile aus, sondern darüberhinaus auch die Werksabgabepreise. Die Artikelkarten sind wie folgt abgelocht:

Spalten 1 bis 3:	Dreistellige Ordnungsnummer; diese wird bei der Sortierung der Karten benötigt. Sie legt somit die Reihenfolge der Karten innerhalb der Ziehkartei fest.
Spalten 4 bis 59:	Beschreibung des Automodells bzw. des Zubehörteiles
Spalte 60:	leer
Spalten 61 bis 67:	Verkaufspreis in der Form 1234.56
Spalten 68 bis 73:	leer
Spalten 74 bis 80:	Werksabgabepreis

Bei den Karten, die für die einzelnen Autos erstellt werden, wird zusätzlich ein Dollarzeichen in die Spalten 60 und 73 gelocht.

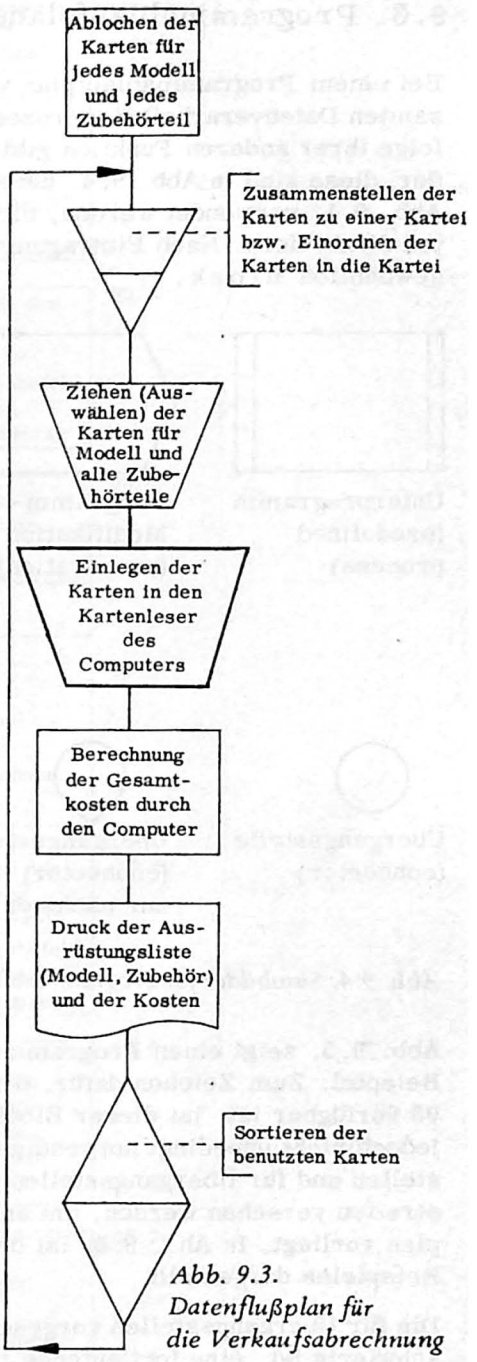
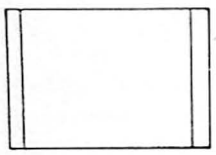


Abb. 9.3. Datenflußplan für die Verkaufsabrechnung

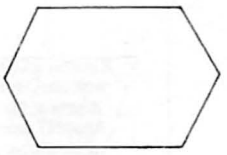
Da jede Kartenspalte in ein Byte eingelesen wird, stehen die Ziffern also mit Zonenteilen im Hauptspeicher. Die Preise müssen deshalb in gepackte Dezimalzahlen umgewandelt werden, bevor sie auf die Gesamtsummen addiert werden können. Nach der Verarbeitung der letzten Karte müssen die Gesamtsummen für die Druckaufbereitung wieder umgewandelt werden, so daß ihre einzelnen Ziffern wieder als Zeichen mit Zonenteilen im Hauptspeicher stehen. Der Datenflußplan, der für diese Aufgabenstellung entwickelt wurde, ist in Abb. 9.3. gezeigt.

9.3. Programmablaufpläne

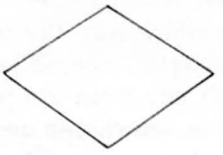
Bei einem Programmablaufplan wird die Betonung auf den speziellen Teil des gesamten Datenverarbeitungsprozesses gelegt, der vom Computer ausgeführt wird. Infolge ihrer anderen Funktion gibt es für Programmablaufpläne einige andere Sinnbilder; diese sind in Abb. 9.4. aufgeführt, sie dürfen zusammen mit den Sinnbildern von Abb. 9.1. verwendet werden, nicht jedoch zusammen mit den in Abb. 9.2. gezeigten Sinnbildern. Nach Eintragung einer Operation in ein Sinnbild nennt man das ganze gewöhnlich Block.



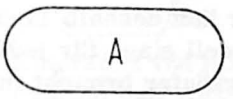
Unterprogramm (predefined process)



Programm-Modifikation (preparation)



Verzweigung (decision)



Grenzstelle (terminal, interrupt) Für A kann z.B. Beginn, Ende, Zwischenhalt usw. geschrieben werden

Übergangsstelle (connector)

Übergangsstelle (connector) zur nächsten Seite

Übergangsstelle (connector) von der vorhergehenden Seite

Abb. 9.4. Sinnbilder für Programmablaufpläne

Abb. 9.5. zeigt einen Programmablaufplan für das im vorigen Abschnitt besprochene Beispiel. Zum Zeichen dafür, daß ein detaillierter Programmablaufplan für den Block 05 verfügbar ist, ist dieser Block durch zwei Randstreifen markiert, eine nützliche, jedoch nicht unbedingt notwendige Methode. Mit Ausnahme der Symbole für Grenzstellen und für Übergangsstellen dürfen alle anderen Symbole mit derartigen Randstreifen versehen werden, um anzudeuten, daß für diese Blöcke ein detaillierter Teilplan vorliegt. In Ab.. 9.6. ist der detaillierte Teilplan für den Block 05 unseres Beispieles dargestellt.

Die für Übergangsstellen vorgesehenen Sinnbilder werden immer dann verwendet, wenn es schwierig ist, eine fortlaufende Linie zur Kenntlichmachung der Ablaufrichtung zu zeichnen. Die Symbole für Übergangsstellen zu anderen Seiten sollten ausschließlich nur für den Übergang von einer Seite zu einer anderen Seite reserviert bleiben, wobei die Pfeilspitze auf die Anschlußseite verweist. Die Sinnbilder für Unterprogramme und Programm-Modifikationen haben wir noch nicht diskutiert; die Symbole sind hier nur der Vollständigkeit wegen aufgeführt.

Der nächste Schritt beim Schreiben eines Programmes befaßt sich mit der Belegung des Hauptspeichers (Speicherplatzzuordnung) und mit dem Aufbau der Eingabe- bzw. Ausgabesätze. Im Abschnitt 9.2. haben wir für unser Beispiel bereits die Einteilung der Lochkarten kennengelernt. Den Ausgabezeilen wollen wir nunmehr fast die gleiche Einteilung

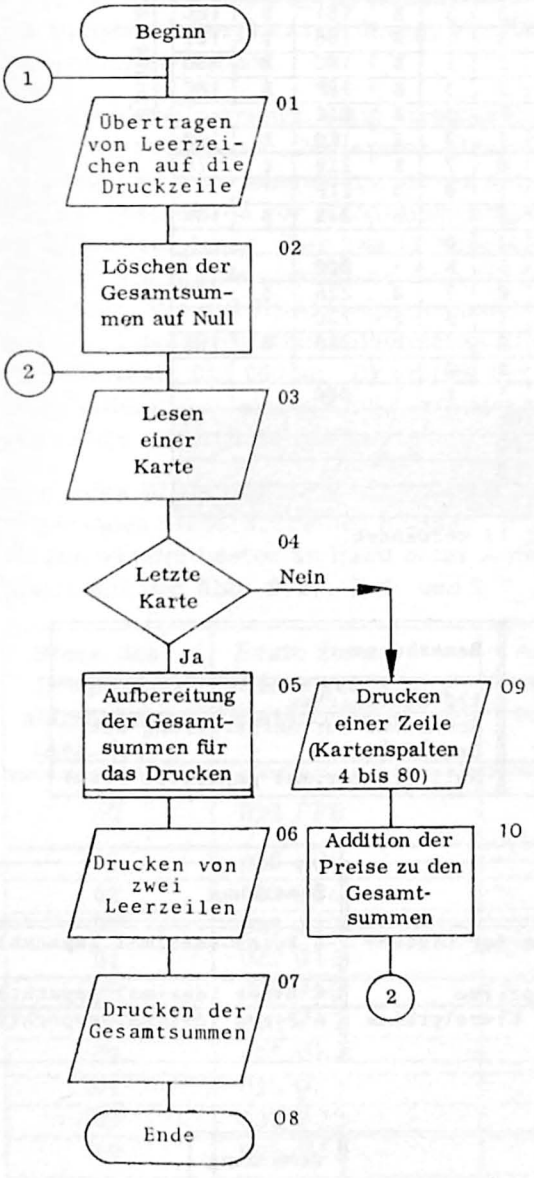


Abb. 9.5. Programmablaufplan für die Verkaufsabrechnung

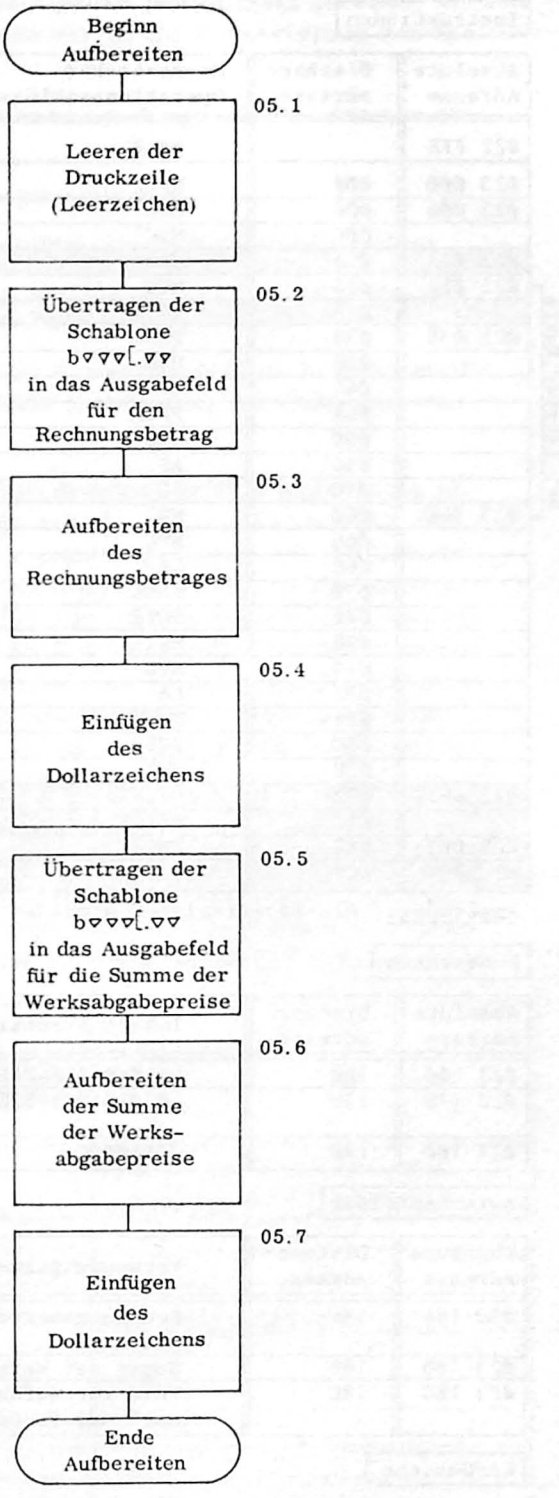


Abb. 9.6. Detaillierter Teilplan für den Block 05 des Programmablaufplanes für die Verkaufsabrechnung

Instruktionen

Absolute Adresse	Distanz-adresse	Mnemonischer Operationsschlüssel	Maschinencode						
			OP						
Ø22 FFE		BALR	Ø5	B	Ø				
Ø23 ØØØ	ØØØ	MVC	D2	7	7	B	1EØ	B	1ØØ
Ø23 ØØ6	ØØ6	MVC	D2	Ø	3	B	184	B	18Ø
	ØØC	MVC	D2	Ø	3	B	188	B	18Ø
Ø23 Ø12	Ø12	READ	Ø2319Ø,Ø23Ø6Ø						
Ø23 Ø22	Ø22	MVC	D2	4	C	B	1E3	B	193
	Ø28	PRINT	Ø231EØ						
Ø23 Ø38	Ø38	PACK	F2	3	6	B	18C	B	1CC
	Ø3E	MVO	F1	2	1	B	18C	B	18C
	Ø44	AP	FA	3	3	B	184	B	18C
	Ø4A	PACK	F2	3	6	B	186	B	1D9
	Ø5Ø	MVO	F1	2	1	B	18C	B	18C
	Ø56	AP	FA	3	3	B	188	B	18C
	Ø5C	BC	47	F	Ø	B	Ø12		
Ø23 Ø6Ø	Ø6Ø	MVC	D2	7	7	B	1EØ	B	1ØØ
	Ø66	MVC	D2	Ø	7	B	21B	B	178
	Ø6C	LA	41	1	Ø	B	21F		
	Ø7Ø	EDMK	DF	Ø	7	B	21B	B	184
	Ø76	BCTR	Ø6	1	Ø				
	Ø78	MVI	92	5	B	1	ØØØ		
	Ø7C	MVC	D2	Ø	7	B	228	B	178
	Ø82	LA	41	1	Ø	B	22C		
	Ø86	EDMK	DF	Ø	7	B	228	B	188
	Ø8C	BCTR	Ø6	1	Ø				
	Ø8E	MVI	92	5	B	1	ØØØ		
Ø23 Ø92	Ø92	PRINT	Ø231ØØ						
	ØA2	PRINT	Ø231ØØ						
Ø23 ØB2	ØB2	PRINT	Ø231EØ						
Ø23 ØC4	ØC2	EOJ							

Bedingter Sprung bei letzter Karte

Unbedingter Sprung

Anmerkung: Als Basisregister wird das Register 11 verwendet.

Konstanten

Absolute Adresse	Distanz-adresse	Inhalt (sedezimal)	Bemerkung
Ø23 1ØØ	1ØØ	4Ø4Ø4Ø4Ø4Ø4Ø4Ø4Ø···4Ø	12Ø Leerzeichen
Ø23 178	178	4Ø2Ø2Ø2Ø214B2Ø2Ø	Schablone zur Aufbereitung der Druckfelder
Ø23 18Ø	18Ø	ØØØØØØØC	Null (in dezimal gepackter Form)

Zwischenfelder

Absolute Adresse	Distanz-adresse	Verwendungszweck	Bemerkung
Ø23 184	184	Rechnungsbetrag (Summe der Listenpreise)	4 Bytes (dezimal gepackt)
Ø23 188	188	Summe der Werksabgabepreise	4 Bytes (dezimal gepackt)
Ø23 18C	18C	Feld zur Aufnahme der Einzelpreise nach dem Packen	4 Bytes (dezimal gepackt)

EA-Gebiete

Absolute Adresse	Distanz-adresse	Verwendungszweck	Bemerkung
Ø23 19Ø	19Ø	Eingabegebiet	8Ø Bytes
Ø23 1EØ	1EØ	Ausgabegebiet (Druckgebiet)	12Ø Bytes

Abb. 9.7. Programm für die Verkaufsabrechnung

zugrundelegen; die Inhalte der Kartenspalten 1 bis 3 sollen jedoch nicht gedruckt werden. Bei der Planung der Hauptspeicherbelegung müssen wir an die Reservierung von Speicherplätzen für folgende Zwecke denken:

a) Speicherplätze für die Maschineninstruktionen unseres Programmes, das wir als Problemprogramm bezeichnen wollen.

b) Speicherplätze für die Konstanten des Problemprogrammes.

c) Speicherplätze für die Eingabe- und Ausgabegebiete, sowie für Zwischenfelder.

d) Speicherplätze für Supervisorroutinen, die zur Ausführung der für unser Beispiel vorgesehenen EA-Operationen herangezogen werden sollen.

Nach reiflicher Überlegung entschließen wir uns zu der in den Abb. 9.7. dargestellten Hauptspeicherplatzzuordnung; wir führen hierbei gleichzeitig die Codierung des Programmes mit auf.

Das Problemprogramm setzt sich aus drei deutlich gegeneinander abgegrenzten Abschnitten zusammen. Den ersten Abschnitt wollen wir als Initialisierung bezeichnen; er wird ausgeführt, bevor die erste Datenkarte gelesen wird. Er besteht aus der Instruktion vor READ und entspricht damit den Blöcken 01 und 02 des Programmablaufplanes. Der zweite Abschnitt wird für alle eingelesenen Karten wiederholt, er stellt eine sogenannte Schleife dar. Zu ihm haben wir die Instruktionen zu zählen, die mit READ beginnen und bis BC reichen; sie entsprechen den Blöcken 09 und 10 des Programmablaufplanes. Der dritte Abschnitt fängt schließlich beim Speicherplatz 023 060 an. Er umfaßt die Blöcke 05, 06, 07 und 08. Da er die durch Feststellung der letzten Karte erforderlichen Schritte umfaßt, kommt er nur einmal zum Zuge (Schlußabschnitt).

Für jeden Block des Programmablaufplanes haben wir eine bestimmte Menge von Instruktionen niederschreiben müssen. Wie unterschiedlich die Mengen sein können, zeigen wir am besten an Hand einer Aufstellung (Abb. 9.8.). Vergleichen wir ihren Inhalt mit den Abb. 9.5., 9.6. und 9.7., so finden wir ihre Aussage leicht bestätigt.

Block des Programmablaufplanes (Abb. 9.5.)	Erste zum Block gehörende Instruktion	Anzahl der erforderlichen Instruktionen	Bemerkung
00	022 FFE	1	Laden des Basisregisters (nicht im PAP[1]) enthalten)
01	023 ØØØ	1	
02	023 ØØ6	2	
03	023 Ø12	1	
04	023 Ø12	-	im READ enthalten (Block 03)
05	023 Ø6Ø	11	
06	023 Ø92	2	
07	023 ØB2	1	
08	023 ØC2	1	
09	023 Ø22	2	
10	023 Ø38	6	

Abb. 9.8. Anzahl der Instruktionen bei den einzelnen Blöcken des PAP[1]) von Abb.

1) PAP ist die Abkürzung von Programmablaufplan.

Wenn Programmierer detaillierte Programmablaufpläne erstellen, die also etwa dem in der Abb. 9.6. dargestellten entsprechen, so neigen einige von ihnen dazu, direkt die einzelnen Instruktionen, freilich meist in einer leicht lesbaren und verständlichen Form, in die Blöcke hineinzustellen. Andere wiederum begnügen sich mit einer Fassung, die nicht bis zu den einzelnen Instruktionen hinunter geht.

9.4. Unterprogramme (Unterroutinen)

Bei der Niederschrift der meisten Programme ergeben sich insofern Schwierigkeiten, weil bestimmte Folgen von beinahe identischen Operationen mehr als einmal im Programm erscheinen müssen, wenn das Problem einer Lösung zugeführt werden soll. Nach dem Einlesen einer Karte müssen wir z. B. den Inhalt zweier Felder in gepackte Dezimalzahlen umwandeln, bei der Ausgabe gepackte Dezimalzahlen aufbereiten, d. h. als Zahlen mit Zonenteilen darstellen. Wir haben uns mit derartigen Problemen schon öfters auseinandergesetzt; jedes Mal haben wir die erforderlichen Umwandlungen nacheinander codiert. Um einen anderen Weg kennenzulernen, mit dem das gleiche erreicht werden kann, wollen wir uns zunächst einmal ein kleines Beispiel ansehen, mit dem die Aufbereitung gepackter Dezimalzahlen erledigt werden kann.

Die Instruktionsfolge von Abb. 9.7., die beim Speicherplatz 023 066 beginnt und beim Speicherplatz 023 091 endet, wollen wir durch die in Abb. 9.9. vorgestellte ersetzen. Außerdem soll eine weitere Instruktionsfolge auf die Speicherplätze 023 0E8 bis 023 0FF gelegt werden.

	Absolute Adresse	Distanzadresse	Mnemonischer Operationsschlüssel	Maschinencode
Hauptprogramm	Ø23 Ø66	Ø66	LA	41 3 Ø B 184
		Ø6A	LA	41 4 Ø B 21B
		Ø6E	BAL	45 E Ø B ØE8
		Ø72	LA	41 3 Ø B 188
		Ø76	LA	41 4 Ø B 228
		Ø7A	BAL	45 E Ø B ØE8
		Ø7E	BC	47 F Ø B Ø92
Unterprogramm	Ø23 ØE8	ØE8	MVC	D2 Ø 7 4 ØØØ B 178
		ØEE	LA	41 1 Ø 4 ØØ4
		ØF2	EDMK	DF Ø 7 4 ØØØ 3 ØØØ
		ØF8	BCTR	Ø6 1 Ø
		ØFA	MVI	92 5 B 1 ØØØ
		ØFE	BCR	Ø7 F E

Abb. 9.9. Unterprogramm zur Aufbereitung gepackter Dezimalzahlen und Aufruf derselben im Hauptprogramm

Wir haben eine Instruktionsmenge, auch Routine genannt, niedergeschrieben, die die gewünschte Aufbereitung leistet, und wir haben diese Routine auf die Hauptspeicherplätze gelegt, die mit der Adresse beginnen, die sich aus dem Inhalt des Basisregisters 11 und der Distanzadresse 0E8 zusammensetzt. Immer, wenn wir diese Routine benutzen wollen, müssen wir die Adresse des aufzubereitenden Feldes ins Register 3 laden und ins Register 4 die Adresse desjenigen Feldes stellen, das die aufbereitete Zeichenfolge aufnehmen soll. Anschließend ist auf die Adresse der ersten Instruktion der Aufbereitungsroutine zu verzweigen. Die dort vorhandene Folge von Instruktionen

erledigt die gesamte Aufbereitung und liefert dabei den aufbereiteten Wert auf dem Feld ab, dessen Adresse im Register 4 an die Routine übergeben wurde. Die letzte Instruktion der Aufbereitungsroutine ist ein Rücksprung zur nächsten Adresse des Programmes, von dem aus die Routine angesprungen wurde. Im Register 14 wurde diese Adresse festgehalten, wofür die BAL-Instruktion verantwortlich war.

Auf die Lösung des gestellten Problemes führen uns bei diesem modifizierten Beispiel zwei Teilprogramme. Sie sind gewissermaßen Programmbausteine eines umfassenden Gesamtprogrammes. Zur Bezeichnung von Programmbausteinen verwendet man auch die Begriffe Programmeinheit oder Modul. Derjenige Modul, mit dem das Gesamtprogramm gestartet wird, heißt Hauptprogramm. Angesprungene Moduln werden Unterprogramme genannt. Man sagt für den Ansprung eines Unterprogrammes auch Aufruf. Unter diesem Aspekt gesehen, gibt es also aufrufende und aufgerufene Programmeinheiten. Auf unseren Fall die soeben eingeführten Begriffe übertragen heißt das: Das Programm, das auf dem Speicherplatz mit der Distanzadresse 000 beginnt, ist also das Hauptprogramm oder Hauptroutine; die ab 0E8 beginnende Instruktionsmenge ist demnach das Unterprogramm (oder auch als Unterroutine bezeichnet). Die Instruktion BAL dient im Hauptprogramm dazu, das Unterprogramm aufzurufen; sie wird deshalb als Verbindungsinstruktion bezeichnet, auch Aufrufinstruktion genannt.

Bei der Darstellung von Programmabläufen werden für Unterprogrammaufrufe spezielle Sinnbilder verwendet (siehe auch Abb. 9.4.).

Das hier behandelte Unterprogramm ist ziemlich trivial. In der Tat braucht man für dieses Unterprogramm mehr neue Instruktionen als man einsparen kann. Außerdem ist es wesentlich komplizierter als eine einzige lineare Instruktionsfolge. Ein Unterprogramm ist meist in sich selbst abgeschlossen, so daß es von jedem beliebigen anderem Programm aufgerufen werden kann. Die Argumente oder auch die Adressen der Argumente werden vorzugsweise in Register gestellt, und die Instruktionen BAL oder BALR dienen dazu, das Unterprogramm aufzurufen, d.h. die Verbindung zwischen dem Hauptprogramm und dem Unterprogramm herzustellen. Unterprogramme zur Berechnung von Quadratwurzeln, Logarithmen usw. sind von so großer Wichtigkeit für den Programmierer, daß zur Lösung dieser Probleme besondere Standardunterprogramme geschrieben wurden, die allgemein zur Verfügung stehen. Wenn immer ein Programmierer die Benutzung einer solchen Funktion wünscht, braucht er deshalb nur noch das Standardunterprogramm in sein Programm einzubauen. Nach welcher Methode z.B. die Errechnung der Quadratwurzel erfolgt, braucht den Programmierer dabei gar nicht zu interessieren; er hat nur das Argument bereitzustellen (d.h. die Zahl, aus der die Quadratwurzel gezogen werden soll), die Verbindungsinstruktionen zu schreiben und dafür Sorge zu tragen, daß das Unterprogramm im Hauptspeicher gespeichert wird. Durch die Bereitstellung von vorhandenen oder zuvor vorbereiteten und ausgetesteten Unterprogrammen kann sich der Programmierer voll auf andere Teile seiner Aufgabenstellung konzentrieren.

Bei dem Eingabe/Ausgabe-Steuerungssystem (Input/Output Control System, abgekürzt IOCS), das wir später behandeln werden, handelt es sich z.B. um eine ganze Reihe von Unterprogrammen, die alle Eingabe- bzw. Ausgabefunktionen abdecken, die normalerweise erforderlich sind. Die SVC-Instruktion wird benutzt, um die Verbindung zu Supervisor- und EA-Unterprogrammen herzustellen, während die beiden Instruktionen BAL bzw. BALR für den Aufruf von Problemunterprogrammen herangezogen werden.

9.5. Dezimalpunkte

Bei den Instruktionen der Dezimalarithmetik und der dualen Festpunktarithmetik ist es nachteilig, daß sie nicht die Stellung des Dezimalpunktes berücksichtigen. Wenn wir z.B. die Instruktion

```
AP |     FA 74 A 09F A 089
```

codieren, so addiert der Computer das siebenstellige Feld mit der Adresse (A) + 089 auf das dreizehnstellige Feld mit der Adresse (A)+09F, wobei angenommen wird, das beide Felder ganze Zahlen enthalten. Dem Computer steht kein Mittel zur Verfügung, durch das er in der Lage wäre, sich die Anzahl der Dizimalstellen zu merken. Somit ist es Sache des Programmierers, selbst dafür Sorge zu tragen, daß die in Additions-, Subtraktions- bzw. Vergleichsoperationen eingehenden Zahlen vor der Durchführung der arithmetischen Operationen ausgerichtet werden, damit die Operationen stellengerecht ablaufen können. Um die hierbei anzustellenden Überlegungen zu demonstrieren, wollen wir in einem Programm die Werte der Funktion

$$f(x) = 3x^3 - 4x^2 + 2x - 13$$

im Intervall von x=2.0 bis x=2.1 mit der Schrittweite von x=0.0001 berechnen lassen. Zu allererst werden wir bei der Vorbereitung der Programmierung unsere gegebene Funktionsgleichung umformen, so daß wir erhalten

$$f(x) = [(3x - 4)\,x + 2]\,x - 13.$$

Die Berechnung des Funktionswertes erfordert nach dieser Umformung nunmehr nur noch drei Multiplikationen gegenüber sechs in der ursprünlichen Form. Nachdem wir somit eine für die Rechnung günstigere Funktionsgleichung bekommen haben, zeichnen wir uns einen Programmablaufplan auf (Abb. 9.10.).

Da f(2)=-1.0 und f(2,1)=+1.3 ist, müssen wir mindestens eine Stelle vor dem Dezimalpunkt vorsehen. Da außerdem für die Abszissenwerte (x-Werte) vier Dezimalstellen berücksichtigt werden müssen, müssen wir für die Berechnung der Funktionswerte 12 Dezimalstellen bereitstellen: Praktisch können wir jedoch diese übermäßige Genauigkeit für die f(x)-Werte niemals gebrauchen. Wir wollen uns deshalb dafür entscheiden, die berechneten Werte auf sechs Dezimalstellen genau auszugeben.

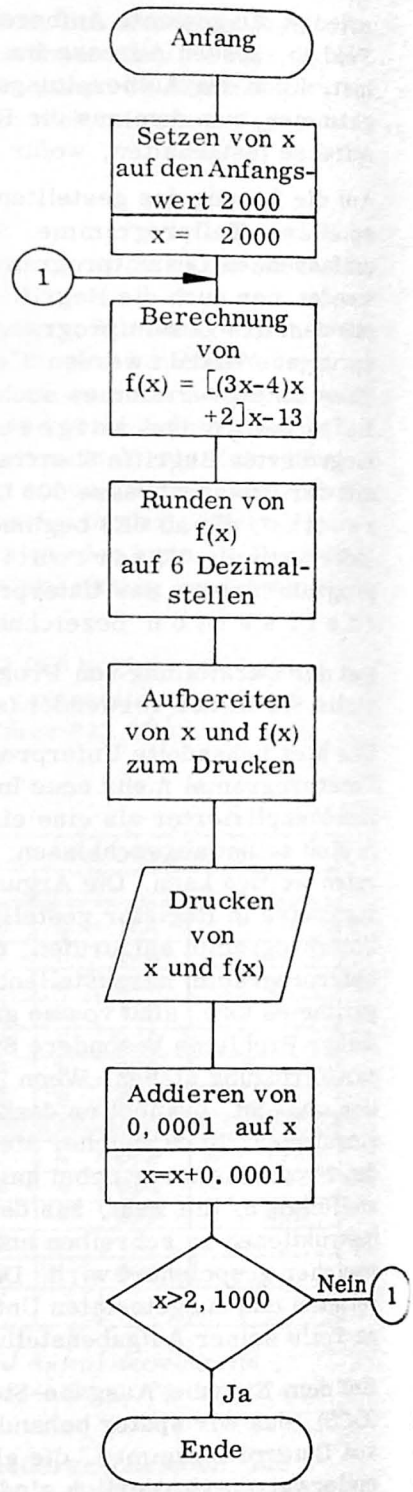


Abb. 9.10. Programmablaufplan

Die Codierung unseres Programmes, verbunden mit einer Übersicht über die Belegung des Hauptspeichers, ist in Abb. 9.11. dargestellt. Das Programm möge auf dem Speicherplatz 023 FFE beginnen. Als Basisregister wird das Register 10 benutzt und durch die BALR-Instruktion mit der Adresse 024 000 geladen.

Instruktionen Beginn: Ø23 FFE

Distanz-adresse	Mnemonischer OP	Maschinencode
	BALR	Ø5 AØ
ØØØ	MVC	D2 Ø2 A ØB2 A ØAC
ØØ6	ZAP	F8 72 A ØB5 A ØB2
ØØC	MP	FC 7Ø A ØB5 A Ø85
Ø12	SP	FB 72 A ØB5 A Ø86
Ø18	MP	FC 7Ø A ØB5 A ØB2
Ø1E	AP	FA 74 A ØB5 A Ø89
Ø24	MP	FC 7Ø A ØB5 A ØB2
Ø2A	SP	FB 77 A ØB5 A ØE8
Ø3Ø	MVI	92 ØC A ØBA
Ø34	AP	FA 51 A ØB5 A Ø96
Ø3A	MVN	D1 ØØ A ØB9 A ØBC
Ø4Ø	MVC	D2 13 A ØCØ A Ø98
Ø46	ED	DE 13 A ØCØ A ØB2
Ø4C	BC	47 AØ A Ø54
Ø5Ø	MVI	92 6Ø A ØCB
Ø54	PRINT	Ø24ØCØ
Ø64	AP	FA 2Ø A ØAC A Ø84
Ø6A	CP	F9 22 A ØAC A ØAF
Ø7Ø	BC	47 CØ A ØØ6
Ø74	EOJ	

Konstanten Beginn: Ø24 Ø84

Distanz-adresse	Inhalt (sedezimal)	Bemerkung
Ø84	1C	Schrittweite von x (Inkrement)
Ø85	3C	3.Ø
Ø86	4Ø ØØ ØC	4.ØØØØ
Ø89	2Ø ØØ ØØ ØØ ØC	2.ØØØØØØØØ
Ø8E	Ø1 3Ø ØØ ØØ ØØ ØØ ØØ ØC	13.ØØØØØØØØØØØØØ
Ø96	Ø5 ØC	Ø.5Ø
Ø98	4Ø 4Ø 21 4B 2Ø 2Ø 2Ø 2Ø 22 4Ø 2Ø 2Ø 21 4B 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø	Schablone für die Druckaufbereitung
ØAC	2Ø ØØ ØC	Anfangswert von x
ØAF	21 ØØ ØC	Endwert von x

Zwischenfelder Beginn: Ø24 ØB2

Distanz-adresse	Verwendungszweck	Länge
ØB2	x (dezimal gepackt)	3 Bytes
ØB5	f(x) (dezimal gepackt)	8 Bytes

Ausgabegebiet Beginn: Ø24 ØCØ

Distanz-adresse	Verwendungszweck	Länge
ØCØ	Zeile für die Druckausgabe	12Ø Bytes

Abb. 9.11. Programm zur Berechnung der Funktionswerte der Funktion $f(x) = 3x^3 - 4x^2 + 2x - 13$

Unser in Abb. 9.11. vorgestelltes Programm haben wir durch Adressen gekennzeichnet, die relativ zum Inhalt des Registers 10 (sedezimal) sind, d.h. durch die Distanzadressen, auch als relative Adressen bezeichnet. Begonnen haben wir mit unseren Adreßangaben nach der einleitenden BALR-Instruktion.

Hinsichtlich der im Programm vorgesehenen Ausrichtung der Dezimalpunkte haben wir noch einige Anmerkungen zu geben. Zunächst übertragen wir den x-Wert in das Ergebnisfeld (f)x auf Speicherplatz (A)+0B5. Der somit erhaltene Inhalt des Ergebnisfeldes besitzt natürlich wie x genau vier Dezimalstellen. Nach der Multiplikation mit 3 bleiben uns die vier Dezimalstellen erhalten, so daß wir ohne weiteres 4.0000 subtrahieren können. Infolge der nächsten Multiplikation mit x wächst unser Zwischenergebnis auf acht Dezimalstellen an, so daß wir nunmehr eine 2 mit acht Dezimalstellen, also 2.00000000, addieren müssen. Nach der letzten Multiplikation mit x hat sich unser Zwischenergebnis auf 12 Dezimalstellen ausgedehnt, so daß wir der letzten zu subtrahierenden Konstanten, nämlich der Zahl 13, ebenfalls 12 Dezimalstellen geben müssen.

Nachdem diese Rechengänge erledigt sind, kann unser 13stelliges Resultat positiv oder negativ sein. Für die vorgesehene sechsstellige Ausgabe muß es nun aber auf die sechste Dezimalstelle gerundet werden. Um eine positive Zahl auf diese Stelle zu runden, muß man eine 5 auf die siebente Dezimalstelle addieren, zur Rundung einer negativen Zahl ist die Subtraktion einer 5 von der siebenten Dezimalstelle erforderlich. Diese Problematik haben wir ohne Abfrage auf eine einfache, doch zugleich elegante Weise gelöst. Ohne das Vorzeichen zu prüfen, haben wir zunächst ein Pluszeichen auf die zehnte Dezimalstelle übertragen (MVI auf 030) und anschließend die gepackte Dezimalzahl 050C, also eine positive, auf den Inhalt des so verkürzten Ergebnisfeldes addiert. Zuletzt haben wir endlich das richtige Vorzeichen von seiner ursprünglichen Stelle auf die neue Vorzeichenstelle des um ein weiteres Byte verkleinerten Feldes übertragen. Damit haben wir zugleich die durch die Rundung überflüssigen Stellen vernachlässigt. Um diesen Vorgang zu veranschaulichen, wollen wir einmal unsere Methode für x=2.0001 eingehend schildern. Für f(x) finden wir zunächst als Ergebnis 000.997799859997-.

Nach Einfügung des Pluszeichens erhalten wir:

00	0.9	97	79	98	0+	99	7-

Auf den Inhalt des auf diese Weise verkürzten Feldes addieren wir stellengerecht eine 5, so daß folgender Vorgang abläuft:

000.99779980+	997-
+ 050+	
000.99780030+	997-

Die rechten vier Halbbytes des gesamten Feldes bleiben also von dieser Operation völlig unberührt. Zum Schluß übertragen wir noch das Minuszeichen auf seinen richtigen Platz, so daß der Inhalt des f(x)-Feldes folgende endgültige Gestalt annimmt:

00	0.9	97	80	0-	0+997-

Wir haben also das rechte Ende unseres ursprünglichen Ergebnisfeldes zerstört. Das schadet aber nichts, da die rechten drei Bytes ohnedies keine Bedeutung für das gewünschte

gerundete Endresultat besitzen. Den erhaltenen Wert 000.997800- bereiten wir anschließend auf und drucken ihn aus.

Fürs Runden von Zahlen können wir natürlich auch andere Wege einschlagen. Wir können z.B. mit Hilfe der BC-Instruktion eine Verzweigung durchführen und das Runden negativer und positiver Zahlen in getrennten Zweigen vornehmen. Es ist jedoch meist vorteilhafter, Programme zu schreiben, die so wenige Verzweigungen aufweisen, wie es nur irgendwie möglich ist.

Wir hätten unser Problem natürlich auch mit Hilfe der dualen Festpunktarithmetik lösen können, doch hätten wir uns dann der AL- bzw. der SL-Instruktionen bedienen müssen, da die Stellenzahl des Resultates das Arbeiten mit Doppelwörtern notwendig gemacht hätte.

Aufgabengruppe 21

1. Fertige für den Block 02 (Abb. 9.5.) einen detaillierten Programmablaufplan an!
2. Zeichne einen Programmablaufplan für folgende Aufgabenstellung: Es sind n und r einzulesen; danach ist $K=(1+r)^n$ zu berechnen. Ausgegeben werden sollen n, r und K. Durch K ist bekanntlich der Kapitalendwert bestimmt, auf den 1.- DM nach n Jahren bei $p=100(r-1)$ Prozent Zinsen anwächst.
3. Entwirf einen Programmablaufplan für ein Unterprogramm, das eine vierstellige ganze Dezimalzahl in eine fünfstellige ganze Oktalzahl konvertiert. Jede Oktalziffer soll dabei ein Byte belegen.
4. Zeichne einen Programmablaufplan für die Aufgabe 4 der Aufgabengruppe 9!
5. Zeichne einen Programmablaufplan für die Aufgabe 6 der Aufgabengruppe 10!
6. Zeichne einen Programmablaufplan für die Aufgabe 11 der Aufgabengruppe 10!
7. Zeichne einen Programmablaufplan für die Aufgabe 4 der Aufgabengruppe 15!
8. Zeichne einen Programmablaufplan für die Aufgabe 4 der Aufgabengruppe 17!
9. Zeichne einen Programmablaufplan für die Aufgabe 6 der Aufgabengruppe 18!
10. Schreibe ein Unterprogramm zur Berechnung von n! (n Fakultät), d.h. zur Berechnung des Produktes $1 \cdot 2 \cdot 3 \cdots (n-1) \cdot n$. Der Wert von n ist hierbei auf den Wertebereich $1 \leqq n \leqq 99$ zu beschränken.
11. Schreibe das im Abschnitt 9.5. besprochene Programm neu und verwende dabei die Instruktionen der dualen Festpunkt-Arithmetik! Beginne mit der Anfertigung von detaillierten Programmablaufplänen!

9.6. Datenmengen

Eine Sammlung von Daten wird Datenmenge [1] genannt. Eine von einem Programm benutzte bzw. angesprochene Datenmenge heißt speziell Datei [2]. Beispiele typischer Datenmengen sind:

1) Im angloamerikanischen Sprachraum als data set bezeichnet.
2) Im angloamerikanischen Sprachraum als file bezeichnet.

a) Namen und Adressen der Abonnenten einer Zeitschrift,
b) die Lohnlisten der XYZ-Gesellschaft,
c) die Gas- und Stromrechnungen der Familie Schöpps in Hintertupfenstein,
d) ein Programm für die IBM /370,
e) eine Sammlung von Programmen für die IBM /370
f) die 1969 von einer Bank eingelösten Barschecks.

Im allgemeinen werden durch ein Programm eine oder mehrere Datenmengen bearbeitet, um eine oder mehrere neue oder abgeänderte Datenmengen zu erzeugen. Typische Eingabedatenmengen befinden sich auf solchen Datenträgern, wie Lochkarten, Magnetbändern oder Plattenspeichern. Ausgabedatenmengen werden auf Datenträgern wie Lochkarten, Plattenspeichern und Druckern erscheinen, um bei einigen repräsentativen Beispielen zu bleiben. Wenn eine aufdatierte Version einer Datenmenge, z.B. von fälligen Kundenrechnungen erzeugt wird, so spricht man von einer neuen Generation der betreffenden Datenmenge.

Einige Aufmerksamkeit müssen wir der Verfügbarkeit von Datenmengen schenken, die auf Magnetbändern oder Plattenspeichern gespeichert sind. So könnte z.B. nach der Aufstellung einer neuen Liste von fälligen Kundenrechnungen die vorhergehenden vergessen, wenn nicht gar vernichtet werden. Bei Steuern dürfte in so einfacher Weise nicht vorgegangen werden; angenommen, das Finanzamt hätte bereits eine Steuerschuldenliste für Juni zusammengestellt, so wäre dennoch die Mailiste nach wie vor von eminenter Bedeutung. Das Problem, wie lange eine Datenmenge aufbewahrt werden muß, wird oft durch Angabe einer Aufbewahrungsfrist gelöst. Das geschieht entweder durch die Nennung des Tages innerhalb eines Jahres, an dem die Frist abläuft, oder durch die Festlegung der Dauer, ausgedrückt in der Zahl von Tagen, der Aufbewahrung.

Eine Datenmenge, die aus einzelnen Gliedern besteht, die selbst wieder durch eindeutige Namen voneinander unterschieden werden können, heißt untergliederte Datenmenge oder auch Bibliothek. Eine ganz gewöhnliche Volksbücherei stellt letzten Endes ebenfalls eine solche Datenmenge dar: Ihre Glieder sind die Bücher und jedes Buch ist durch die Verzeichnisnummer eindeutig identifizierbar, z.B. durch

Q32
Sm43
Copy 4

Eine Volksbücherei führt in der Regel ein sorgfältiges Verzeichnis ihrer Bücher.

Die Bibliotheken von Computerinstallationen enthalten u.a. ausführungsbereite Programme. Ein Bibliotheksverzeichnis hält fest, wo die einzelnen Programme auf den externen Speichern gespeichert sind. Bei einem System IBM /370 befinden sich die Bibliotheken stets auf einer SEDZ [1]. Nachdem ein Programm in eine Bibliothek aufgenommen und die entsprechende Eintragung ins Bibliotheksverzeichnis gemacht wurde, gilt es als katalogisiert. Nicht nur Programme, sondern auch andere Datenmengen können katalogisiert werden. Die Tatsache, daß eine Datenmenge nicht katalogisiert ist, bedeutet nicht etwa, daß der Computer nicht zu ihr zugreifen kann, sie bedeutet nur, daß die Maschinenbedienung die Datenmenge lokalisieren muß, wenn sie von einem Programm angefordert, d.h. benötigt wird.

1) Unter SEDZ versteht man eine Speichereinheit mit direkter Zugriffsmöglichkeit. Hinter dem logischer Begriff „Einheit" verbergen sich, physikalisch gesehen, externe Speichergeräte, die den direkten Zugriff zu den auf ihnen gespeicherten Informationen gestatten, wie Magnetplattenspeichergeräte, Magnettrommelspeichergeräte usw.

9.7. Die Blockung von Sätzen

Wir wollen einmal annehmen, daß auf ein Magnetband 30000 Sätze geschrieben werden sollen; jeder Satz umfasse 320 Bytes. Bei einer Zeichendichte von 1600 Bytes/Zoll [1] belegt jeder Satz gerade 0,2 Zoll (0,508 cm); hinzu kommt die nachfolgende Kluft von 0,6 Zoll (1,524 cm). Somit ergibt sich eine Gesamtlänge von 0,8 Zoll (2,032 cm). Für 30000 Sätze sind also 2000 Fuß (60960 cm) erforderlich, also fast eine ganze Normalrolle von 2400 Fuß (731,5 m) Länge. Von diesen 2000 Fuß entfallen 500 Fuß (15240 cm) auf Daten, also nur 25%, der Rest auf die Klüfte (1500 Fuß=45720 cm).

Unsere nächste Berechnung soll auf folgender Basis aufbauen: Zwei Datensätze sollen gleichzeitig, zusammengefaßt also, auf das Magnetband geschrieben werden. Eine solche physische Zusammenfassung von (logischen) Sätzen wollen wir als physischen Satz oder Block bezeichnen. Diese Annahme reduziert die Zahl der Sätze von 30000 auf 15000; jedoch sind die Sätze nunmehr 640 Bytes lang zu machen. Der gesamte Platzbedarf auf dem Magnetband schrumpft damit auf 1000 Fuß (30480 cm), wovon jeweils die Hälfte (500 Fuß=15240 cm) auf Daten bzw. auf Klüfte entfallen. Faßt man fünf logische Sätze zu einem Block zusammen, reduziert sich der Magnetbandbedarf auf 800 Fuß (24384 cm). Besteht ein Block aus 10 Sätzen, so sind 650 Fuß (19812 cm) auf dem Magnetband bereitzuhalten, bei Blöcken aus 100 Sätzen immerhin noch 515 Fuß (15697,2 cm). Das Zusammenfassen von logischen Sätzen zu physischen Sätzen wird als Blockung bezeichnet.

Für diesen Vorteil der Blockung, der sich in einem geringeren Datenträgerbedarf niederschlägt, haben wir in zweierlei Hinsicht auch zu zahlen. Zuerst müssen wir im Hauptspeicher ein EA-Gebiet (Eingabegebiet beim Lesen, Ausgabegebiet beim Schreiben) bereitstellen, das groß genug ist, um 1,2,5,10 oder sogar 100 logische Sätze aufnehmen zu können. Bei 100 logischen Sätzen sind dafür immerhin 32000 Bytes erforderlich; das halte man sich vor Augen. Nur eine gut ausgerüstete Computerinstallation kann ein so großes reales Hauptspeichergebiet für die EA einer einzigen Datei verkraften. So große Blockungsfaktoren sind also im allgemeinen nicht opportun. Weiterhin ist zu berücksichtigen, daß beim Übergang von einem Blockungsfaktor von 10 zu einem solchen von 100 nur noch etwa 20 % des Magnetbandbedarfs eingespart werden, dagegen zusätzlich rd. 29000 Bytes an realem Hauptspeicherplatz benötigt werden, was einem prozentualen Mehrbedarf von 800 % entspricht. Wir müssen,diese Betrachtungen zeigen das ganz klar auf, die Ersparnisse an Magnetbandplatz und an EA-Zeiten gegen den bereitzustellenden Hauptspeicherbedarf abwägen. Im allgemeinen werden wir freilich die Sätze so weit wie möglich blocken, wobei wir das Limit beachten müssen, das uns durch den zur Verfügung stehenden Hauptspeicherplatz gesetzt ist.

Einige häufig verwendete Begriffe sollen an dieser Stelle nicht unerwähnt bleiben. Die Anzahl der logischen Sätze, die in einem physischen Satz enthalten ist, heißt Blockungsfaktor. Die Länge eines physischen Satzes wird als Blockumfang [2] bezeichnet. Ist der Blockungsfaktor gleich 1, so spricht man oft von ungeblockten Sätzen, bei einem Blockungsfaktor, der größer als 1 ist, von geblockten Sätzen.

Wir können uns auch einen logischen Satz denken, dessen Satzlänge größer als der Blockumfang ist. Das bedeutet, daß ein Block nur einen Teil eines Satzes aufnehmen kann und daß mehr als ein Block für jeden Satz benötigt wird. Der Teil des Satzes, der in einem Block enthalten ist, wird Segment genannt. Vom Satz selbst sagt man,

1) 1600 Bytes/Zoll ≙ 630 Bytes/cm
Für Bytes/Zoll ist die Abkürzung bpi gebräuchlich

2) Auch als Blocklänge bezeichnet

daß er Blöcke überspannt. Dementsprechend wollen wir in solchen Fällen von Spannsätzen reden.

Das zweite Problem, das beim Blocken logischer Sätze zu beachten ist, liegt in der internen Programmverwaltung. Wenn wir beispielsweise einen aus fünf logischen Sätzen bestehenden Block lesen, so müssen wir selbst genau verfolgen und dementsprechend auch beachten, welchen von den fünf Sätzen wir gerade verarbeiten. Eine Methode besteht darin, daß wir ein Register als Basisregister benutzen und in diesem die Adresse des momentan verarbeiteten Satzes festhalten. Nach vollzogener Verarbeitung eines Satzes müssen wir den Registerinhalt um die Satzlänge erhöhen und den so erhaltenen neuen Inhalt mit einem Grenzwert vergleichen. Bei Überschreitung des Grenzwertes wissen wir, daß es Zeit ist, den nächsten Block zu lesen und das Basisregister auf seinen niedrigsten Wert zurückzusetzen. Eine andere Methode besteht darin, daß wir jeden Satz nacheinander in ein Arbeitsgebiet übertragen, so daß bei der Verarbeitung jeden Satzes alle Hauptspeicheradressen gleich sind. Nachdem ein Satz verarbeitet worden ist, übergeben wir die Steuerung einer Verwaltungsroutine, die den nächsten Satz in das Arbeitsgebiet stellt. Wenn es erforderlich ist, muß die gleiche Verwaltungsroutine freilich auch für das Lesen eines neuen Blockes vom Magnetband

Satzart	*Anzahl der EA-Gebiete*	*Besonderes Arbeitsgebiet*	*Überlappung*
Ungeblockt	1	Nein	Überlappung nur bei gepufferten Geräten. Keine Überlappung bei Magnetband- und Plattenspeichereinheiten.
		Ja	Überlappung bei jedem Satz.
	2	Nein	Überlappung bei jedem Satz.
		Ja	Überlappung bei jedem Satz. Die Verwendung eines Arbeitsgebietes bringt keinen Vorteil.
Geblockt	1	Nein	Keine Überlappung.
		Ja	Überlappung beim letzten Satz eines jeden Blockes.
	2	Nein	Überlappung bei jedem logischen Satz.
		Ja	Überlappung bei jedem logischen Satz. Die Verwendung eines Arbeitsgebietes bringt keinen Vorteil.

Abb. 9.12. Überlappung zwischen EA und Verarbeitung bei ungeblockten und geblockten Sätzen

sorgen. Bei Spannungssätzen, die bekanntlich über mehrere Blöcke hinwegreichen können, ist die Benutzung eines Arbeitsgebietes verpflichtend, da das Eingabegebiet in einem bestimmten Augenblick niemals einen ganzen Satz enthält.

Bei Einsatz eines Arbeitsgebietes kann unmittelbar nach dem Übertragen des letzten Satzes eines Blockes ins Arbeitsgebiet der nächste Block bereits vom Band gelesen werden, während zur gleichen Zeit der Satz im Arbeitsgebiet verarbeitet wird. Auf diese Weise überlappen sich Verarbeitung und Lesen. Bei einer Verarbeitung der Sätze im Eingabegebiet ist ein solches Vorgehen nicht möglich.

Ein dritter Lösungsweg bietet sich an, wenn die Verarbeitung im Eingabegebiet stattfindet, jedoch ein zweites Eingabegebiet bereitgestellt wird und abwechselnd einmal in das eine und dann in das andere eingelesen wird. Für eine ausgedehntere Überlappung zwischen Eingabe und Verarbeitung ist dafür zwar vorgesorgt, doch erfordert dieses Vorgehen einen höheren Hauptspeicherbedarf und die Bereitstellung eines Registers, das die Basisadresse für das gegenwärtig benutzte Eingabegebiet aufnimmt.

Eine vierte Alternative basiert auf zwei Eingabegebieten und einem Arbeitsgebiet. Sie wird gewöhnlich nicht benutzt, da sie nur dann Zeit einspart, wenn die meisten Sätze nur wenig oder gar keine Verarbeitung erfordern, einige wenige jedoch erhebliche Verarbeitungszeit. Die Abb. 9.12. zeigt, wieweit der Effekt der Überlappung sich unter den verschiedenen Verhältnissen niederschlägt.

9.8. Satzarten

Es ist zweckmäßig, wenn wir uns an dieser Stelle mit den vier verschiedenen Satzarten [1] beschäftigen, die man bei der Datenverarbeitung kennt. Man unterscheidet bei Datenmengen folgende Satzarten:

a) Sätze fester Länge
Alle logischen Sätze besitzen die gleiche Länge. Diese Satzart wird durch den Buchstaben F gekennzeichnet (Fixed Length).

b) Sätze variabler Länge
Die logischen Sätze weisen verschiedene Längen auf. Am Anfang jeden Satzes steht jedoch seine Länge. Kein Satz ist jedoch länger als der Blockumfang. Somit ist der Programmierer jederzeit in der Lage, die Länge der einzelnen Sätze festzustellen. Diese Satzart wird durch den Kennbuchstaben V gekennzeichnet (Varible Length).

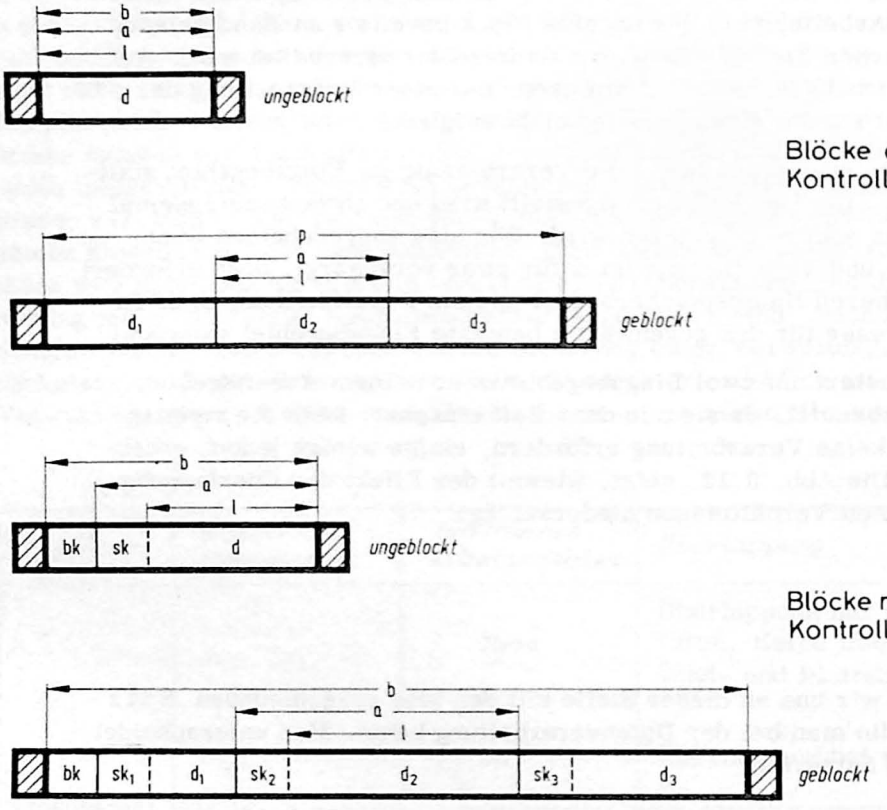
c) Spannsätze variabler Länge
Auch diese Satzart zeichnet sich durch Sätze aus, die variabel lang sind. Einige Sätze können jedoch eine Länge aufweisen, die größer als der Blockumfang ist; sie überspannen also einen Block. Diese Satzart wird durch den Buchstaben S gekennzeichnet. (Spanned variable Length).

d) Sätze unbestimmter (undefinierter) Länge
Die Sätze sind vollkommen variabel in ihrer Länge. Es gibt keine Standardfestlegung bezüglich ihrer Länge. Diese Satzart wird durch den Kennbuchstaben U gekennzeichnet (Undefined Length).

---

1) Auch als Satzformate bzw. Satzformen bezeichnet.

Bevor wir auf die einzelnen Satzarten bezüglich der Speicherung ihrer Sätze auf den externen Speichern eingehen, wollen wir uns zuerst einmal mit der Form der Speicherung befassen. In Abb. 9.13. wollen wir die generellen Probleme veranschaulichen.



Es bedeuten:

b	Block (physischer Satz)
a	Aufzeichnungssatz
l	logischer Satz oder Segment (bei Spannsätzen)
bk	Blockkontrollfeld
sk	Satzkontrollfeld
d	Daten

Anmerkung: Klüfte sind durch ▨ gekennzeichnet

Abb. 9.13. Beispiele für die Speicherung von Sätzen

Abb. 9.13. macht uns einmal mit den drei wichtigen Begriffen vertraut, auf die wir bei der Speicherung achten müssen. Auf das externe Speichermedium, den Datenträger, wird stets der physische Satz oder Block aufgezeichnet. Bei gewissen Satzarten geht diesem Block ein Blockkontrollfeld voraus, in der Abb. mit bk bezeichnet. Aufgezeichnet werden stets die sogenannten Aufzeichnungssätze, entweder als Teil des Blockes (bei geblockten Sätzen) oder allein im Block (bei ungeblockten Sätzen). Bei bestimmten Satzarten enthält der Aufzeichnungssatz außer dem logischen Satz mit den Daten noch ein Satzkontrollfeld, in der Abb. mit sk bezeichnet. In einem Programm brauchen nur die logischen Sätze aufgebaut bzw.

betrachtet zu werden, wenn man sich an die Konventionen eines Betriebssystemes und der in ihm enthaltenen Datenmanagementroutinen hält. Sicher könnte man sich auch andere Aufzeichnungsformen ausdenken als die in Abb. 9.13. dargestellten, doch sollten wir uns beizeiten an diejenigen gewöhnen, die bei den für die IBM /370 entwickelten Betriebssystemen und den in sie integrierten Datenmanagementroutinen gelten. In den nachfolgenden Betrachtungen wollen wir uns speziell mit den Aufzeichnungsformen bei den verschiedenen Satzarten beschäftigen.

Mit Sätzen fester Länge läßt sich am einfachsten arbeiten; sie sollten deshalb immer dann verwendet werden, wenn es nur irgendwie möglich ist.

Für Sätze variabler Länge und für Spannsätze ist ein Übereinkommen erzielt worden, das darin besteht, jedem logischen Satz ein vier Bytes umfassendes Satzkontrollfeld voranzustellen. In den ersten beiden Bytes wird die Länge des Aufzeichnungssatzes, d.h. die um 4 vergrößerte Länge des logischen Satzes bzw. des Segmentes (bei Spannsätzen) festgehalten. Da zwei Bytes (d.h. 16 Dualstellen ohne Vorzeichenstelle) einen Wertebereich von 1 bis 65536 einschließen, ist damit zwar die Satzlänge der Aufzeichnungssätze auf 65536 Bytes begrenzt. Ein so hoher Grenzwert bedeutet für die Praxis keinerlei Einschränkungen. Die beiden letzten Bytes enthalten auf ihren 16 Dualstellen, wenn Sätze des V-Formats vorliegen. Bei Spannsätzen muß jedoch außer der Länge des Satzes oder Segmentes auch erkennbar sein, welches Segment vorliegt, wenn ein Satz mehrere Blöcke überspannt, bzw. ob es sich um einen nicht in Segmente zerlegten Satz handelt. Zu diesen Informationen wird das dritte Byte des Satzkontrollfeldes herangezogen, das vierte Byte enthält auch hier lauter duale Nullen. Es sind folgende Verschlüsselungen im dritten Byte gebräuchlich:

00	Der logische Satz zerfällt nicht in Segmente
01	Es liegt das erste Segment eines in Segmente zerlegten logischen Satzes vor
02	Es liegt das letzte Segment eines in Segmente zerlegten logischen Satzes vor
03	Es liegt ein mittleres Segment eines in Segmente zerlegten logischen Satzes vor

Alle Verschlüsselungen sind in sedezimaler Form angegeben. Um die Bedeutung der Segmenttypen zu erfassen, wollen wir einen Blick auf die Abb. 9.14. werfen, die uns vier aufeinanderfolgende Blöcke einer Datei zeigt, die aus Sätzen im SB-Format besteht.

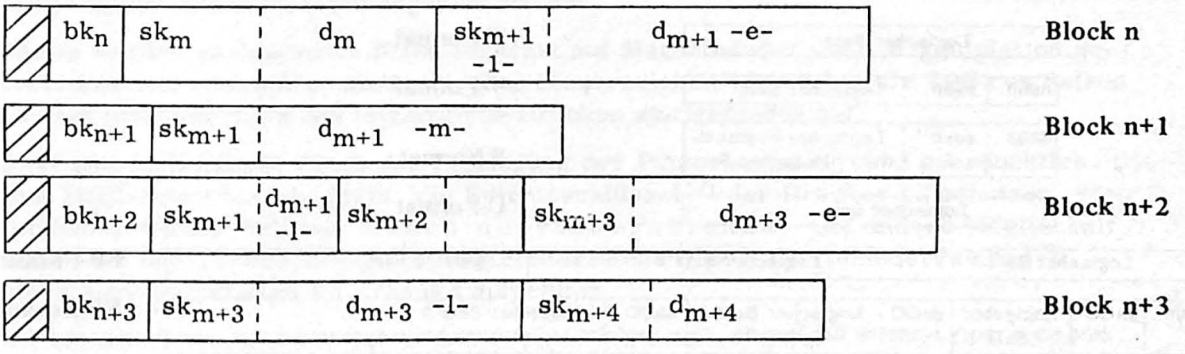


Abb. 9.14. Beispiel für die Blockbelegung bei Spannsätzen

Wie üblich sind durch bk_i mit i=n, n+1, n+2, n+3 die Blockkontrollfelder gekennzeichnet. Die Symbole sk_j mit j=m, m+1, m+2, m+3, m+4 markieren die Satzkontrollfelder von fünf in den vier dargestellten Blöcken enthaltenen logischen Sätze d_j. Durch e wird das erste Segmen t, durch m ein mittleres Segment und durch l das letzte Segment eines logischen Satzes bezeichnet. Wir ersehen also aus unserem Beispiel, daß die logischen Sätze d_m, d_{m+2} und d_{m+4} nicht in Segmente zerlegt sind, da sie gerade noch in einen Block hineinpassen. Bei den logischen Sätzen d_{m+1} und d_{m+3} ist das nicht der Fall. Sie werden in Segmente aufgetrennt und auf verschiedene Blöcke verteilt: d_{m+1} in drei, d_{m+3} in zwei Segmente. Die Satzkontrollfelder der Aufzeichnungssätze enthalten hier also die Kontrollinformationen für die Segmente, die als Aufzeichnungssätze auf dem Datenträger stehen.

Jedem Block variabler Länge und jedem Block mit Spannsätzen variabler Länge wird am Anfang ein weiteres vier Bytes langes Kontrollfeld, das sogenannte Blockkontrollfeld, hinzugefügt, das die Blocklänge auf seinen ersten beiden Bytes enthält. Die letzten beiden Sätze unbestimmter Länge werden wie Sätze fester Länge aufgezeichnet, d.h. bei ihnen fehlen die Längenangaben. Sie werden stets als ungeblockt angesehen. Blocken und Entblocken liegen allein in der Verantwortung des Programmierers. Der Gebrauch von Sätzen undefinierter Länge sollte bei Magnetbändern tunlichst vermieden werden.

Wir wollen jetzt in einer Übersicht alle Kennungen zusammenfassen, die es für die verschiedenen Satzarten gibt (Abb. 9.15.).

Kennung	Satzart
F	Sätze fester Länge, ungeblockt
FB	Sätze fester Länge, geblockt
V	Sätze variabler Länge, ungeblockt
VB	Sätze variabler Länge, geblockt
S	Spannsätze variabler Länge, ungeblockt
SB	Spannsätze variabler Länge, geblockt
U	Sätze undefinierter Länge, ungeblockt

Abb. 9.15. Kennungen der verschiedenen Satzarten

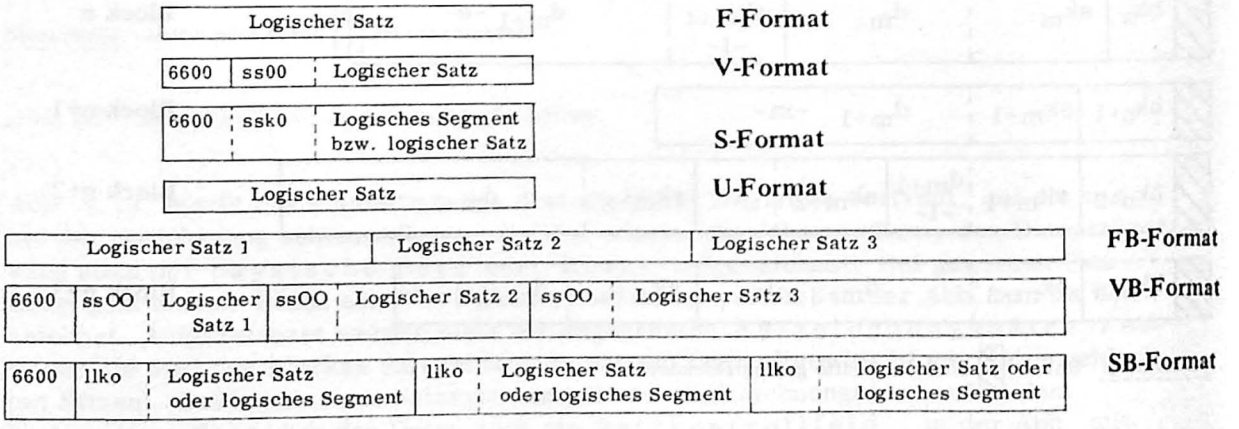


Abb. 9.16. Satzarten

Bei geblockten Sätzen ist es immer möglich, daß der letzte Block kürzer ist als die vorhergehenden, d.h. er enthält nicht die sonst mögliche Zahl von Sätzen.

Bei der Festlegung von Eingabe- bzw. Ausgabegebieten ist es allgemeiner Usus, daß diese an Doppelwortbegrenzungen beginnen, wenn das F-, FB- oder das U-Format vorliegt. Bei den anderen vier Satzarten wird üblicherweise eine Ausrichtung der EA-Gebiete auf Wortgrenzen vorgenommen.

In Abb. 9.16. ist noch einmal die Form der Blöcke bei den sieben verschiedenen Satzarten dargestellt. Wir haben in dieser Abbildung folgende Abkürzungen benutzt:

ss	Zwei Bytes langes Teilfeld mit der dualen Satzlänge
bb	Zwei Bytes langes Teilfeld mit der dualen Blocklänge
00	Zwei Bytes langes Teilfeld, dessen 16 Bitstellen sämtlich 0 enthalten
0	Ein Byte langes Teilfeld, dessen 8 Bitstellen sämtlich 0 enthalten
k	Ein Byte langes Teilfeld, das die Segmentkennung enthält (bei Sätzen im S- bzw. SB-Format)

Zwischen der Blocklänge l und den Satzlängen m_i der $i=1,2,\cdots,n$ Sätze eines Blockes besteht folgender Zusammenhang:

$$l = 4 + \sum_{i=1}^{n} m_i$$

Es sei daran erinnert, daß die Satzlängen m_i auch das 4 Byte lange Längenfeld mit einschließen.

Zur Abb. 9.16. wäre noch zu bemerken, daß wir den Blockungsfaktor 3 unterstellt haben.

9.9. Steuerzeichen für Drucker

Sind Ausgabedaten dazu bestimmt, einmal gedruckt [1] zu werden, so wird üblicherweise jedem Satz eine Information hinzugefügt, die die gewünschte Papierbewegung reguliert. Gemäß Übereinkommen geschieht das dadurch, daß ein Druckersteuerzeichen in das erste Byte jeden Satzes im F-, FB- und U-Format bzw. in das fünfte Byte jeden Satzes (d.h. unmittelbar nach dem vier Bytes umfassenden Längenfeld) im V-, VB-, S- und SB-Format hineingestellt wird. Dieses Steuerzeichen wird selbst nicht gedruckt. Der Satzumfang ist jedoch um 1 zu erhöhen, denn die Satzlänge muß der Hinzufügung des Steuerzeichens angepaßt werden.

Häufig werden zu druckende Sätze zunächst auf Magnetbänder oder Magnetplatten geschrieben und erst später gedruckt. Das Steuerzeichen wird dabei als Teil des Satzes solange mitgeführt,bis das tatsächliche Drucken stattgefunden hat.

Zwei unterschiedliche Arten der Festlegung der Papierbewegung sind gebräuchlich. Die eine Möglichkeit besteht darin, die Befehlsschlüssel [2] der Drucker zu benutzen. Man spricht in diesem Fall von Maschinensteuerzeichen. Die andere Möglichkeit beruht auf der Verwendung genormter Steuerzeichen, der ASCII-Steuerzeichen [3]. Diese sind geschlossen im Anhang I aufgeführt.

---

1) Wir meinen damit, daß das Drucken nicht unmittelbar erfolgen muß, sondern daß durchaus zuvor noch eine Zwischenspeicherung der Ausgabedaten auf elektromagnetischen Datenträgern (Magnetbändern oder Magnetplatten) erfolgen kann, bevor das eigentliche Drucken erfolgt. Veranlassung hierzu gibt der relativ langsame Druckvorgang.

2) Befehlsschlüssel der Kanalbefehlswörter (CCW_S).

3) ASCII ist die Abkürzung von American Standard Code for Information Interchange.

Charakteristisch für die ASCII-Steuerzeichen ist es, daß sie die Papierbewegung vor dem Drucken implizieren. Das aber heißt konsequenterweise, daß zwei Befehle zum Drucker hinausgeschickt werden müssen. Das erste CCW vollführt die Papierbewegung, das zweite veranlaßt den eigentlichen Druckvorgang bei Unterdrückung von Vorschub bzw. Zeilentransport. Die Verwendung von Maschinensteuerzeichen erlaubt eine gewisse Überlappung der für die Papierbewegung erforderlichen Zeiten und der Druckzeiten. Im Schnitt resultiert daraus eine um etwa 5 % höhere Druckgeschwindigkeit als bei Benutzung der ASCII-Steuerzeichen.

Um anzuzeigen, daß Steuerzeichen Teil des Satzes sind, wird bei Verwendung von Maschinensteuerzeichen der Buchstabe M, bei Verwendung der ASCII-Steuerzeichen der Buchstabe A an die Satzartkennung (siehe Abb. 9.13.) angehängt. Somit bedeutet die Satzformatangabe FBA, daß geblockte Sätze fester Länge vorliegen, an deren erster Stelle jeweils ein ASCII-Steuerzeichen steht. Wir erinnern: Ein Satzumfang von 121 Stellen führt auf eine 120 stellige Druckzeile.

Steuerzeichen werden auch für die Ablagefachauswahl eingesetzt, wenn Lochkarten gestanzt werden sollen. Auch hier könnnen sowohl Maschinensteuerzeichen als auch ASCII-Steuerzeichen benutzt werden.

9.10. Bandkennsätze (Bandetiketten)

Erfolgt die Eingabe von Daten mit Lochkarten oder auf Belegen, so kann sich der Programmierer diese stichprobenweise ansehen, um sicherzugehen, daß er für ein bestimmtes Programm die richtigen Datenträger verwendet. Anders ist das jedoch bei Magnetbändern, wenn diese als Datenträger verwendet werden, denn die auf ihnen aufgezeichneten Informationen sind nicht lesbar. Nun kann es vorkommen, daß bei einer Organisationsabteilung mehrere Tausend Magnetbandrollen lagern und benutzt werden. Es ist deshalb klar, daß man nach irgendeiner Methode suchen muß, die eine gewisse Sicherheit gegenüber der mißbräuchlichen Benutzung von Magnetbändern bietet. Überwiegend geschieht das durch das Aufzeichnen spezieller Sätze, der sogenannten K e n n s ä t z e oder E t i k e t t e n , an beiden Rollenenden sowie zu Beginn und Ende einer Datenmenge. Am Anfang aufgezeichnete Sätze heißen V o r s ä t z e oder A n f a n g s e t i k e t t e n , am Ende aufgezeichnete N a c h s ä t z e oder E n d e t i k e t t e n. Im Grunde genommen, kann eigentlich jeder diese Methode so anwenden, wie es ihm gefällt. Es sind jedoch gewisse Standardformate für die Kennsätze erdacht worden, deren Verwendung infolge der Austauschfähigkeit und der Einheitlichkeit der Informationen so große Vorteile bietet, daß evtl. Nachteile als völlig untergeordnet empfunden werden müssen. Hält man sich jedenfalls an die Regeln des Aufbaus dieser sogenannten Standardkennsätze, so kann man sich große Mühen ersparen. - Im restlichen Teil dieses Buches werden wir über die verschiedenen Programme sprechen, die von der IBM geliefert werden, um die Programmierung der IBM /370 zu erleichtern. Alle diese Programme sehen die Verwendung von Magnetbändern mit Standardkennsätzen vor. Es gibt folgende verschiedene Arten von Kennsätzen:

1. Datenträgerkennsätze (Spulenetiketten)
   Es gibt zwei verschiedene Arten von Datenträgerkennsätzen:
   a) Datenträgervorsätze (Volume Labels)
      Die Datenträgervorsätze sind immer am Anfang einer jeden Magnetbandrolle aufgezeichnet. Wir werden für sie grundsätzlich die Abkürzung VOL gebrauchen.

b) Datenträgernachsätze (End of Volume Labels)
Diese Kennsätze werden auf das Ende eines Datenträgers, d.h. einer Magnetbandrolle aufgezeichnet. Belegt jedoch eine Datei mehrere Magnetbandrollen, so wird ein solcher Kennsatz nicht auf das Ende der letzte Rolle geschrieben. Wir werden für die Datenträgernachsätze grundsätzlich die Abkürzung EOV gebrauchen.

2. Datenmengenkennsätze (Datenmengenetiketten)
Es gibt zwei verschiedene Arten von Datenmengenkennsätzen:

a) Datenmengenvorsätze (Header Labels)
Diese Kennsätze stehen vor jeder Datenmenge, die auf ein Magnetband aufgezeichnet ist. Wir werden für sie grundsätzlich die Abkürzung HDR gebrauchen.

b) Datenmengennachsätze (Trailer Labels)
Datenmengennachsätze stehen stets hinter den Datenmengen, die auf Magnetbändern aufgezeichnet sind. Wir werden für sie grundsätzlich die Abkürzung EOF (End of File) verwenden.

Bevor wir im einzelnen die Kennsätze diskutieren, wollen wir uns um die Klärung einiger Begriffe bemühen. Unter einem Datenträger verstehen wir eine physikalische Einheit, auf die Informationen aufgezeichnet werden können. Beispiele hierfür sind Magnetbandrollen und Plattenstapel sowie Lochkarten, Lochstreifen usw. Unter einer Datenmenge (siehe Abschnitt 9.6.) wollen wir eine Menge aufeinanderbezogener Sätze verstehen. Mitunter wird hierfür auch der Begriff Datenbestand [1] benutzt. Datenmengen können einen Teil eines Datenträgers, einen Datenträger oder mehrere Datenträger belegen. Datenmengen können bei der Eingabe, bei der Ausgabe oder auch bei der Eingabe und bei der Ausgabe angesprochen werden; für das betreffende Programm gelten sie dann als Datei. Sind die Informationen einer Datenmenge aufeinanderfolgend angeordnet, d.h. die Sätze sind bzw. werden in der gleichen Reihenfolge aufgezeichnet wie sie verarbeitet werden, so spricht man von einer starr fortlaufend oder konsekutiv organisierten Datenmenge. Datenmengen auf Magnetbändern, Karten oder auf Druckseiten können naturgemäß nur starr fortlaufend organisiert sein.

Alle Bandkennsätze sind grundsätzlich 80 Bytes lang. Die werden durch den Inhalt ihrer ersten vier Stellen auseinandergehalten. Wir wollen diesen Inhalt als Kennung bezeichnen. Der in der nachstehenden Aufstellung aufgeführte Buchstabe n steht für eine der Ziffern 1 bis 8. Es gibt folgende Kennungen:

VOLn	Datenträgervorsatz	*standardisierte Kennsätze*
EOVn	Datenträgernachsatz	
HDRn	Datenmengenvorsatz	
EOFn	Datenmengennachsatz	
UHLn	Benutzervorsatz	*nicht standardisierte Kennsätze*
UTLn	Benutzernachsatz	

Neben den standardmäßig vorhandenen Datenmengenvorsätzen können Benutzer außerdem auch eigene Datenmengenvorsätze angeben und verwenden; wir wollen diese allgemein einfach als Benutzervorsätze bezeichnen (Datenträgervorsätze sind immer standardmäßig, der Benutzer kann eigene nicht angeben und verwenden). Wir werden für die Benutzervorsätze grundsätzlich die Abkürzung UHL gebrauchen. Die aufge-

1) Der Begriff Datei (englisch: file) sollte man nur auf diejenigen Datenmengen beziehen, die von bestimmten Programmen angesprochen, d.h., entweder gelesen oder geschrieben werden, die also unter allen vorhandenen Datenmengen eine ausgezeichnete Rolle einnehmen.

zeichneten Benutzervorsätze enthalten auf ihren ersten vier Stellen UHLn. Die Abkürzung UHL kommt vom englischen Begriff „User Header Label".

Spezielle Benutzernachsätze für Datenmengen enthalten stets die Zeichenfolge UTLn in ihren ersten vier Bytes. Wir selbst werden für sie immer die Abkürzung UTL gebrauchen; diese aus dem Englischen stammende Abkürzung kommt von „User Trailer Label".

Die Kennsätze sind von den Datenblöcken durch eine sogenannte Bandmarke, auch Bandbegrenzungszeichen genannt, getrennt. Das Ende der verwertbaren Informationen auf einem Magnetband ist durch zwei aufeinanderfolgende Bandmarken gekennzeichnet.

Die Abb. 9.17. zeigt die Reihenfolge der Kennsätze, wenn eine Datenmenge einen ganzen Datenträger belegt. Zuerst kommen die Datenträgervorsätze, dann die Datenmengenvorsätze. Auf diese folgen evtl. Benutzervorsätze und eine Bandmarke. Nach der Datenmenge steht ebenfalls eine Bandmarke; dieser wiederum folgen die Datenmengennachsätze und die Benutzernachsätze. Zum Schluß signalisieren zwei Bandmarken das Ende der auf dem Magnetband aufgezeichneten Informationen. Bandmarken sind in der Abb. 9.17., 9.18. und 9.19. durch BM angedeutet.

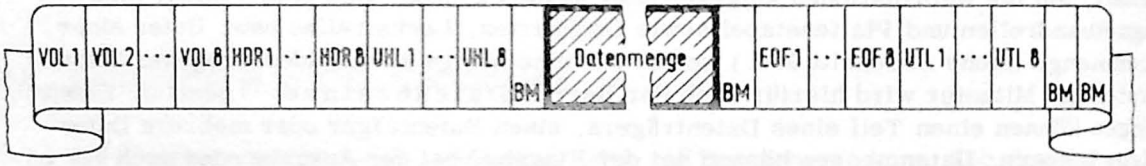


Abb. 9.17. Magnetbandrolle mit einer Datenmenge (schematisch)

In der Abb. 9.18. wird die Anordnung der Kennsätze für den Fall gezeigt, daß ein Datenträger von mehreren Datenmengen belegt ist. Zuerst kommen die Datenträgervorsätze, danach die Standardvorsätze und evtl. die Benutzervorsätze für die erste Datenmenge. Diese Vorsätze werden durch eine Bandmarke begrenzt. Nach der Bandmarke ist die erste Datenmenge aufgezeichnet. Dieser schließen sich eine Bandmarke, die Standardnachsätze und evtl. vorhandene Benutzernachsätze für die erste Datenmenge an. Hinter allen Nachsätzen folgt erneut eine Bandmarke. Danach kommen die von einer Bandmarke begrenzten Vorsätze für die zweite Datenmenge, die zweite Datenmenge selbst, eine Bandmarke, die Nachsätze der zweiten Datenmenge und eine Bandmarke. Für weitere gespeicherte Datenmengen beginnt das Wechselspiel von neuem. Eine Bandmarke wird nach dem letzten Kennsatz jeder Datenmenge geschrieben; eine Ausnahme macht jedoch die letzte Datenmenge: dieser schließen sich nämlich zwei aufeinanderfolgende Bandmarken an.

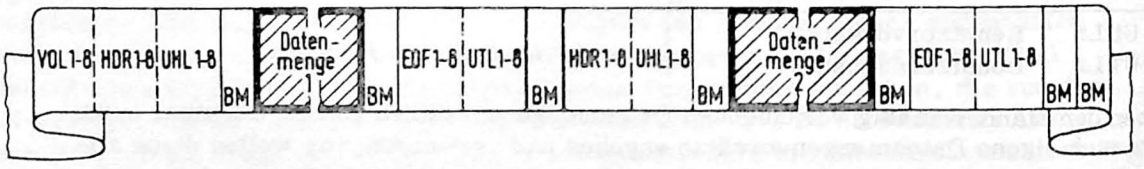


Abb. 9.18. Magnetbandrolle mit mehr als einer Datenmenge (schematisch)

In Abb. 9.19. wird die Anordnung der Kennsätze auf den verschiedenen Rollen gezeigt, wenn eine Datenmenge mehr als eine Magnetbandrolle belegt. Zu Beginn der ersten Bandrolle erscheinen die VOL-, HDR- und UHL-Kennsätze in der üblichen bekannten Reihenfolge. Am Ende dieser Rolle wird eine Bandmarke geschrieben. Dieser folgen die Datenträgernachsätze, denen sich Benutzernachsätze anschließen können. Abgeschlossen wird

die erste Rolle durch eine Bandmarke. Jede zusätzlich von der Datenmenge belegte Magnetbandrolle beginnt mit den gleichen Kennsätzen wie die erste Bandrolle; erst nach diesen werden die zur Datenmenge gehörenden Informationen fortgesetzt. Mit Ausnahme der letzten Bandrolle sind am Schluß jeder Bandrolle die gleichen Informationen aufgezeichnet wie bei der ersten Bandrolle. Bei der letzten von der Datenmenge belegten Bandrolle treten die Datenmengennachsätze an Stelle der Datenträgernachsätze; außerdem wird sie durch zwei aufeinanderfolgende Bandmarken abgeschlossen. Ansonsten ist, das sei noch einmal betont, die letzte Bandrolle genau so aufgebaut wie alle anderen.

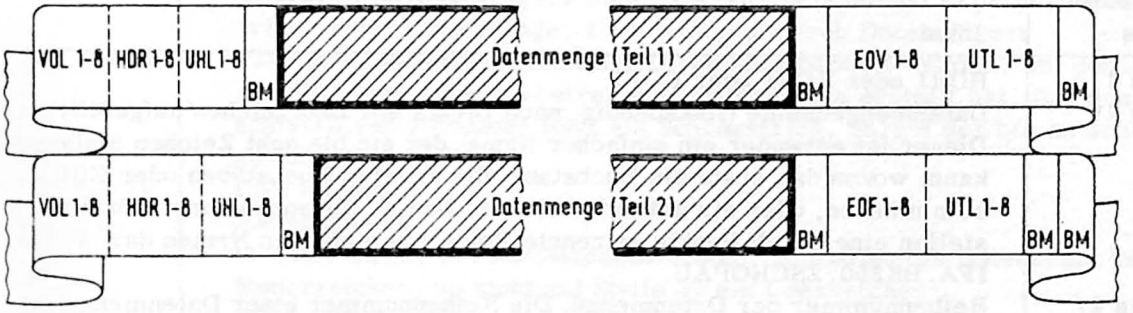


Abb. 9.19. Datenmenge auf mehreren Rollen; dargestellt sind 1. und letzte Rolle (schematisch)

Die standardmäßige Verwendung von Kennsätzen ist nach folgenden Richtlinien Notwendig sind nur die Kennsätze VOL1, HDR1, EOF1 und EOV1; die Verwendung aller anderen Kennsätze ist freigestellt. Sie werden in der Praxis gewöhnlich auch weggelassen. Mitunter werden höchstens noch die Kennsätze HDR2, EOV2 und EOF2 benutzt.

Mit Ausnahme dieser vier unbedingt erforderlichen Kennsätze können alle anderen Kennsätze, von einigen wenigen zwingend vorgeschriebenen Informationen abgesehen, beliebig zusammengestellt werden. Jeder Kennsatz muß jedoch eine Länge von 80 Bytes besitzen und in den ersten drei Bytes eine der Zeichenfolgen VOL, HDR, EOF, EOV, UHL oder UTL aufweisen. Im vierten Byte muß schließlich die fortlaufende Nummer der betreffenden Kennsatzart stehen. Es sind grundsätzlich nicht mehr als acht Kennsätze einer bestimmten Art erlaubt; der Benutzer kann jedoch weniger als acht vorsehen. Es ist höchst ungewöhnlich, alle für einen Datenträger oder für eine Datenmenge möglichen Kennsätze zu verwenden; gerade der Anfangsprogrammierer ist bestens beraten, wenn er auf alle nichterforderlichen Kennsätze verzichtet.

In den nachfolgenden Ausführungen wollen wir den Aufbau aller unbedingt erforderlichen Kennsätze kennenlernen.

a) Datenträgervorsatz VOL1

Bytes	Inhalt
1 bis 4	VOL1
5 bis 10	Reihennummer des Datenträgers. Diese Nummer wird einer Bandrolle gewöhnlich beim Erwerb derselben gegeben und zur Kontrolle auch auf die Außenseite des Spulenkörpers geschrieben. Diese Ordnungsnummer darf durchaus alphanumerische Zeichen enthalten, doch wird sie gewöhnlich nur aus Ziffern, z. B. 000186, bestehen.
11	Sicherheitsinformation. Besitzt dieses Byte den Wert 0, so ist keine weitere Identifizierungsinformation erforderlich, wenn das Band verwendet

	werden soll. Steht dagegen in diesem Byte eine 1, so ist für die Benutzung des Bandes eine weitere Identifizierung notwendig (Paßwort).
12 bis 21	Leerzeichen
22 bis 31	Leerzeichen
32 bis 41	Leerzeichen
42 bis 41	Name und Adresse des Eigentümers
52 bis 80	Leerzeichen

b) Kennsätze HDR1, EOV1 und EOF1

Bytes	Inhalt
1 bis 4	HDR1 oder EOV1 oder EOF1
5 bis 21	Datenmengenname (linksbündig, nach rechts mit Leerzeichen aufgefüllt). Dieser ist entweder ein einfacher Name, der ein bis acht Zeichen umfassen kann, wovon das erste ein Buchstabe, die anderen Buchstaben oder Ziffern sein müssen, oder ein gekennzeichneter Name. Gekennzeichnete Namen stellen eine durch Punkte getrennte Folge von einfachen Namen dar, z.B. IFA.BK350.ZSCHOPAU.
22 bis 27	Reihennummer der Datenmenge. Die Reihennummer einer Datenmenge ist stets gleich der Reihennummer des Datenträgers, auf dem sich die Datenmenge befindet; belegt eine Datenmenge mehrere Datenträger, so steht hier grundsätzlich die Reihennummer des ersten Datenträgers, d.h. des Datenträgers, auf dem die Datenmenge beginnt.
28 bis 31	Laufende Nummer des Datenträgers. Für den ersten Datenträger einer Datenmenge ist diese gleich 0001, für den zweiten 0002 usw.
32 bis 35	Laufende Nummer der Datenmenge. Für die erste Datenmenge auf einem Datenträger ist diese gleich 0001, für die zweite Datenmenge gleich 0002 usw.
36 bis 39	Generationsnummer. Wird eine Datei öfters auf den neuesten Stand gebracht, so sind gewöhnlich mehrere Dateigenerationen vorhanden, da man meist jede Generation über eine gewisse Zeit hinweg aufbewahrt. Zumindest zerstört man kaum eine Generation schon in dem Augenblick, in dem eine neue Generation erzeugt wird. Dieses Feld dient nun zur Identifizierung der einzelnen Generationen einer Datei, beginnend mit 0000, 0001 bis 9999. Besitzt eine Datenmenge keine Generationen, so ist hier 0000 aufgezeichnet.
40 bis 41	Versionsnummer der Generation. Eine ganze Zahl zwischen 01 und 99 hält die einzelnen Stadien bei der Verarbeitung der betreffenden Generation der Datenmenge fest.
42 bis 47	Erstellungsdatum in der Form bjjttt. Hierbei bedeutet b das Leerzeichen; durch jj wird das Jahr und durch ttt der laufende Tag des Jahres angegeben. Beispiel: 67 291, d.h. der 291 Tag des Jahres 1967.
48 bis 53	Verfalldatum in der gleichen Form wie das Erstellungsdatum. Durch dieses Datum ist der erste Tag festgelegt, an dem die betreffende Datenmenge durch eine andere überschrieben werden darf.
54	Sicherheitsinformation. Bei 0 ist keine weitere Identifizierungsinformation erforderlich, bei 1 muß eine solche vorhanden sein (Paßwort).
55 bis 60	000000
61 bis 73	Systemschlüssel. Dieser identifiziert das Programmsystem, das den Kennsatz erzeugte.
74 bis 80	Leerzeichen

c) Kennsätze HDR2, EOV2 und EOF2

Bytes	Inhalt
1 bis 4	HDR2, EOV2 oder EOF2
5	Satzart: F, U oder V. Für Spannsätze wird V verwendet.
6 bis 10	Maximaler Blockumfang (maximale Blocklänge)
11 bis 15	Maximale Satzlänge (bei Sätzen im U. Format steht hier 0)
16	Zeichendichte auf dem Band (2=800bpi, 3=1600bpi, 4=6250bpi) [1]
17	Position des Datenträgers innerhalb der Datenmenge (0 beim ersten Datenträger einer Datenmenge, 1 bei jedem anderen Datenträger)
18 bis 34	Job- und Jobstepidentifikation. Diese kann benutzt werden, um den Job zu identifizieren, der die betreffende Datenmenge erstellt hat. Auf den bis zu achtstelligen Jobnamen folgt ein Schrägstrich, darauf der bis zu achtstellige Jobstepname
35 bis 36	Leerzeichen
37	Steuerzeichen (A beim ASCII, M beim Maschinencode). Enthalten die Datensätze keine Steuerzeichen, so steht auf Stelle 37 ein Leerzeichen
38	Leerzeichen
39	Blockungskennzeichen (B für die Satzarten FB oder VB, S für die Satzart S, R für die Satzart SB, Leerzeichen für die Satzarten F, V und U)
40 bis 80	Leerzeichen

d) Benutzerkennsätze

Bytes	Inhalt
1 bis 4	UHLn oder UTLn
5 bis 80	Beliebige Informationen des Benutzers

9.11. Kennsätze bei SEDZ [2]

Alle Speichereinheiten mit direkter Zugriffsmöglichkeit (SEDZ) erfordern sowohl Datenträgerkennsätze als auch Datenmengenkennsätze. Der VOL-Kennsatz besitzt den gleichen Aufbau wie der VOL-Kennsatz bei Magnetbändern. Bei Magnetbändern sind bekanntlich die Stellen 12 bis 17 dieses Kennsatzes leer, bei SEDZ enthalten sie die Anfangsadressen des Datenträgerinhaltsverzeichnisses (Volume Table of Contents - VTOC).

Der VOL1- Kennsatz steht stets als 3. Satz in der Spur 0 des Zylinders 0. Die Satzbereiche 1 und 2 dieser Spur sind grundsätzlich für Sätze reserviert, die für das einleitende Programmladen (IPL) benötigt werden. Bei den externen Speichergeräten, die als Systemresidenz dienen, sind diese Satzbereiche belegt, bei allen anderen Geräten enthalten sie duale Nullen. Der VOL1-Kennsatz weist ein vier Bytes langes Schlüssel-

1) bpi ist die Abkürzung von Bytes per Inch. Die aufgeführten Werte entsprechen der Reihe nach 315 Bytes/cm, 630 Bytes/cm und 2461 Bytes/cm.

2) SEDZ ist die Abkürzung von Speichereinheit mit direkter Zugriffsmöglichkeit. Hinter diesem Begriff im logischen Sinne verbergen sich diejenigen externen Speichergeräte, bei denen die Möglichkeit des direkten Zugriffs zu freigewählten Datensätzen besteht. Hierzu zählen u. a. Magnetplattenspeicher und Magnettrommelspeichergeräte.

feld auf, dem die Zeichenfolge VOL1 zugewiesen ist. Das Datenfeld selbst ist 80 Bytes lang. Werden andere Datenträgervorsätze (VOL2 bis VOL8) benutzt, so schließen sie sich dem VOL1-Kennsatz an. Notwendig sind jedoch die Vorsätze VOL2 bis VOL8 nicht.

Benutzerkennsätze können in gleicher Weise wie bei den Magnetbändern benutzt werden. Jeder aufgezeichnete Benutzerkennsatz besteht aus einem vierstelligen Schlüsselfeld und einem achtzigstelligen Datenfeld. Je nach Art des Kennsatzes enthält das Schlüsselfeld UHLn oder UTLn(mit n=1, 2, ··· , 8). Werden Benutzerkennsätze gewünscht, so belegen sie die erste Spur desjenigen Speicherraumes, der für die betreffende Datenmenge zur Verfügung steht. Dadurch, daß die Benutzerkennsätze Speicherplätze belegen, die normalerweise den Daten vorbehalten sind, befinden sie sich im Datengebiet und nicht im Kennsatzgebiet der entsprechenden SEDZ.

Das Datenträgerinhaltsverzeichnis (VTOC) setzt sich aus einer Menge von Sätzen zusammen, in denen die Plätze verzeichnet sind, auf denen sich die Datenmengen befinden, die auf dem betreffenden Gerät gespeichert sind.[1] Alle VTOC-Sätze haben ein 44stelliges Schlüsselfeld und ein 96stelliges Datenfeld. Dabei werden mehrere verschiedene Satzformate benutzt. Der erste Satz, auf den der VOL1-Kennsatz hinzeigt, ist ein Satz des Formates 4. Er enthält die Größe des Datenträgerinhaltsverzeichnisses, ausgedrückt durch die höchste und die niedrigste Adresse der von ihm belegten Speicherplätze, sowie die Lage und die Anzahl der unbenutzten Ausweichspuren. Außerdem sind in ihm einige Konstanten über das Gerät enthalten wie die Anzahl der Stellen pro Spur, die Anzahl der Spuren pro Zylinder usw.

Bevor ein neuer Plattenstapel zum ersten Mal eingesetzt wird, wird gewöhnlich ein Dienstprogramm der IBM herangezogen, um die Spuradressen (Hausadressen) und die Spurbeschreibungssätze R_0 aufzuzeichnen und um den Plattenstapel auf defekte Spuren zu überprüfen. Das gleiche Dienstprogramm schreibt auch den Datenträgerkennsatz, der danach niemals mehr zerstört wird, umd legt das Datenträgerinhaltsverzeichnis an, indem es den Satz des Formats 4 sowie einen Satz des Formats 5 schreibt.

Ein Satz des Formats 5 kann benutzt werden, um die Größe der benutzten und unbenutzten Bereiche auf der SEDZ zu registrieren. Im Bedarfsfalle kann mehr als ein Satz dieses Formats geschrieben werden. Jeder verweist dann auf den Folgesatz mittels eines Adreßfeldes. Das Adreßfeld des letzten Satzes ist auf Null gesetzt. Jeder Bereich wird dabei durch ein zehnstelliges Merkfeld definiert. Das erste Byte dieses Feldes zeigt den Inhalt des betreffenden Teiles in folgender Weise an (der Code ist sedezimal angegeben):

Code	Inhalt des Bereiches
01	Die folgenden neun Bytes sind unbenutzt
01	Daten
02	Es liegt ein Bereich mit Überläufen vor (nur bei indexsequentiell organisierten Datenmengen)
04	Es liegt ein Bereich mit Inhaltsverzeichnissen vor (nur bei indexsequentiell organisierten Datenmengen)
40	Bereich mit Daten (mit Ausnahme der ersten Spur, die Kennsätze enthält)
80	Bereich mit Daten, der Zylinder gemeinsam mit anderen Datenmengen benutzt
81	Der Bereich ist aus vollständigen Zylindern zusammengesetzt

1) Die von Datenmengen bzw. von Teilen derselben belegten zusammenhängenden Plätze auf SEDZ werden Bereiche genannt (im angloamerikanischen Sprachraum als Extents bezeichnet).

Einige der in dieser Übersicht aufgeführten Fälle haben wir noch nicht diskutiert. Sie sind deshalb nur im Sinne der Vollständigkeit unserer Aussage zu betrachten.

Das zweite Byte enthält die laufende Nummer des Bereiches; diese ist für den ersten Bereich gleich 0, für den zweiten gleich 1 usw., stets als Dualzahl ausgedrückt.

Die nächsten vier Bytes enthalten ZZKK, d.h. Zylinder Z und Schreib-/Lesekopf K der ersten Spur des Bereiches. In gleicher Weise wird die letzte Spur des Bereiches durch ZZKK auf den letzten vier Stellen notiert. Gewöhnlich belegt eine Datenmenge einen Bereich, der nicht unterteilt ist. Sie kann und darf aber auch so untergebracht werden, daß sie mehrere Bereiche umfaßt. Für einige Anwendungen ist es z.B. wünschenswert, daß man zwei Datenmengen auf der gleichen Gruppe von Zylindern unterbringt, wobei die eine Datenmenge vielleicht die Spuren $0, 1, \cdots$ und die andere die Spuren $\cdots, 18, 19$ belegt. Derartige benutzte Zylinder werden als gemeinsam benutzte Zylinder bezeichnet (siehe sedezimale Verschlüsselung 80) im Byte 1 des Merkfeldes.

Jede Datenmenge besitzt einen Kennsatz des Formats 1. Dieser entspricht dem Datenmengenvorsatz (HDR1-Kennsatz) von Magnetbändern. Das Schlüsselfeld enthält dabei den Datenmengennamen, der aus einfachen Namen zusammengesetzt werden kann. Ein einfacher Name weist 1 bis 8 Zeichen auf, das erste muß ein Buchstabe sein, alle anderen müssen Buchstaben oder Ziffern sein. Ein, zwei, drei, vier oder fünf einfache Namen können zusammengefaßt werden, um den Datenmengennamen zu bilden. Zwischen zwei einfachen Namen muß außerdem ein Punkt stehen. Innerhalb des Datenmengennamens dürfen keine Leerzeichen vorhanden sein. Er muß ferner stets linksbündig angegeben werden. Zwei Beispiele hierzu:

a) SYS73320.T005953.RF000.DREIECK.R0000013
b) HOLLAND.AMSTER.DAM

Normal wird ein Datenmengenname in der soeben beschriebenen Weise geformt. Wenn jedoch die Datenmenge mehrere Generationen umfaßt, so muß der Datenmengenname kurz genug sein, um in das 44stellige Schlüsselfeld noch die Generationsbezeichnung unterbringen zu können. Diese schließt sich an den letzten einfachen Namen über einen Punkt an und sieht wie folgt aus: Auf den Buchstaben G folgt die vierstellige Generationsnummer; sind von der bezeichneten Generation außerdem noch Versionen vorhanden, so folgt auf die Generationsnummer noch der Buchstabe V und die zweistellige Versionsnummer. Somit ist beispielsweise die 12. Version der 324. Generation einer Datenmenge wie folgt zu bezeichnen:

DSN.ABC.G0324V12

Generations- und Versionsnummern sind stets ganze vorzeichenfreie Dezimalzahlen.

Das 96stellige Datenfeld des Kennsatzes vom Format 1 besitzt folgenden Aufbau:

Bytes	Inhalt
1	F1 (sedezimal); Identifikation des Kennsatzformats
2 bis 7	Laufende Nummer der Datenmenge
8 und 9	Laufende Nummer des Datenträgers
10 bis 12	Erstellungsdatum in der Form JTT; für J sind hierbei Werte von 00 bis 99, für TT von 1 bis 366 zulässig, beide als Dualzahlen
13 bis 15	Verfalldatum in der Form JTT (J und TT haben die gleiche Bedeutung wie bei den Bytes 10 bis 12)

16	Anzahl der Bereiche, die die betreffende Datenmenge auf diesem Datenträger beansprucht. Ein Bereich, der nur Benutzerkennsätze enthält, ist hierbei nicht eingeschlossen
17	leer mit Ausnahme bei untergliederten Datenmengen, wobei hier die Anzahl der Bytes spezifiziert ist, die im letzten Block des Adreßverzeichnisses (directory) mit relevanten Informationen versehen sind.
18	stets 00 (sed)
19 bis 31	Systemschlüssel
32 bis 38	stets 00 (sed)
39 bis 40	Sedezimale Verschlüsselung der Organisationsform der Datenmenge 0000 Nicht definiert 0200 Untergliederte Datenmenge 2000 Regional organisierte Datenmenge 4000 Starr fortlaufend organisierte Datenmenge (konsekutive Organisationsform) 8000 Logisch fortlaufend organisierte Datenmenge
41	Satzformat. Die einzelnen Bitstellen dieses Bytes besitzen folgende Bedeutung: 0 und 1 01-Sätze im V-Format 10-Sätze im F-Format 11-Sätze im U-Format 2 1-Spurenüberlauf wird verwendet 3 1-Geblockte Sätze 4 1-Mit Ausnahme des letzten Blockes gibt es keine verkürzten Blöcke in der Datenmenge, vorausgesetzt, auf den Bitstellen 0 und 1 steht der Code 10. Ist dort jedoch der Code 01 verzeichnet, so bedeutet eine 1, daß Spannsätze vorliegen. 5 bis 6 00-Keine Steuerzeichen 01-Steuerzeichen nach ASCII (früher als ASA-Steuerzeichen bekannt) 10-Maschinensteuerzeichen 7 unbenutzt
42	80 (sed), wenn die Schreibgültigkeitsprüfung bei der Erstellung der Datenmenge durchgeführt wurde. Das bedeutet, daß allen Schreiboperationen ein Lesen folgt, das auf die soeben geschriebenen Daten zugreift. Dadurch wird das korrekte Schreiben gesichert.
43 und 44	Maximale Blocklänge (als Dualzahl)
45 und 46	Maximale Satzlänge (als Dualzahl). Bei Sätzen im U-Format steht hier 0, bei Spannsätzen mit möglichen Längen, die größer als 32756 sind, steht hier 8000 (sedezimal).
47	Schlüssellänge (als Dualzahl). Eine 0 zeigt an, daß die Sätze ohne Schlüssel aufgezeichnet sind.
48 und 49	Relative Schlüsselposition [1] innerhalb des Satzes (als Dualzahl). Wenn bei geblockten Sätzen der Schlüssel im Satz enthalten ist, so bedeutet das die Stelle innerhalb des Satzes, auf der der Schlüssel beginnt.
50	Die Bitstellen 0 bis 3 besitzen folgende Bedeutung: 0 1-Letzter Datenträger einer Datenmenge 1 1-Die Datenmenge muß auf denselben absoluten Plätzen der SEDZ verbleiben 2 1- Die Blocklänge muß ein Vielfaches von 2 Bytes sein 3 1- Die Datenmenge ist durch eine Sicherheitsinformation geschützt

1) Im angloamerikanischen Sprachraum als „relative key position", abgekürzt RKP, bezeichnet.

51 bis 54	Mitunter benutzt, um anzuzeigen, wie der Bereich für die Datenmenge errechnet wurde
55 bis 59	Diese fünf Stellen beziehen sich auf den letzten geschriebenen Block (Satz) in einer starr fortlaufend organisierten Datenmenge. Dabei geben die ersten beiden Stellen die relative Nummer derjenigen Spur an, die den letzten Block (Satz) aufgenommen hat (die erste Spur des ersten Bereiches besitzt die relative Spurnummer 0). Die dritte Stelle besagt die laufende Nummer des Blockes (Satzes) in der Spur; damit ist durch den Inhalt der Stellen 55 bis 57 die Stellung des letzten Blockes (Satzes) innerhalb der Datenmenge eindeutig umrissen. Auf den Stellen 58 und 59 ist die Anzahl der Bytes verzeichnet, die auf der durch die Stellen 55 und 56 bezeichenten Spur verbleiben
60 und 61	stets 00 (sed)
62 bis 71	1. Merkfeld
72 bis 81	2. Merkfeld
82 bis 91	3. Merkfeld
92 bis 96	Adreßfeld. Dieses enthält die Adresse des nächsten Kennsatzes für diese Datenmenge in der Form ZZKKS (ZZ Zylinder; KK Schreib-/Lesekopf; S Satz), sofern ein solcher benötigt wird. Anderenfalls enthalten diese Stellen 00 (sed).

Wir wollen hier nicht die Formate der anderen Kennsätze behandeln, da sie ohne Interesse für den Programmierer sind.

Die Benutzung des Spurbeschreibungssatzes R_0 ist gemäß Absprache eingeschränkt. R_0 wird stets mit einer achtstelligen Information beschrieben, die folgenden drei Zwecken dienen kann:

1. Wenn eine Spur schadhaft ist, so werden die ersten vier Bytes dazu verwendet, um in ihnen die Adresse der Ersatzspur (Ausweichspur) zu speichern. Normalerweise werden die Spuren der innersten Zylinder als Ersatzspuren verwendet.
2. Bei regional organisierten Datenmengen enthält R_0 in der Form ZZKKS (Zylinder, Schreib/Lese-Kopf, Satz) zunächst die Adresse des letzten auf diese Spur geschriebenen Satzes, die beiden anschließenden Bytes halten die Anzahl der auf der Spur verbleibenden unbenutzten Bytes fest. Das letzte Byte bleibt unbenutzt.
3. Bei logisch fortlaufend (indexsequentiell) organisierten Datenmengen verweisen die ersten fünf Stellen, d.h. also ZZKKS, auf einen Satz im Überlaufbereich. Die nächsten beiden Bytes geben die Anzahl der Bytes an, die in der Überlaufspur noch frei sind.

In allen anderen Fällen bleibt R_0 unbenutzt. Die Tabellen im Anhang H basieren auf dieser konventionellen Benutzung von R_0.

Aufgabengruppe 22

1) Diskutiere die beiden folgenden Bandkennsätze!

a) HDR1PROBLEM.N1bbbbbbb0000130002000200 34
01b67034b680340bbbbbbSYSTEMPROBbbbbbbbbbb
b) EOV1PROBLEM.N1bbbbbbb0000130002000100 34
01b67034b68034008043SYSTEMPROBbbbbbbbbbbb

Anmerkung: b bedeutet hier das Leerzeichen

2) Für das Magnetband von Aufgabe 1b) ist der VOL1-Kennsatz zusammenzustellen. Unter Eigentümer ist „370TXTCNT" einzutragen.

3) Wenn bei der Aufgabe 1) das Magnetband zusätzlich den Kennsatz

```
HDR2F0175200876 31THREEbbb/REELbbbbbbbBb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
```

besitzt, so ist es möglich, die Datenmenge näher zu beschreiben. Welche Aussagen können zusätzlich gemacht werden?
Anmerkung: b bedeutet hier das Leerzeichen

4) Wenn die Datenmenge von Aufg. 3) auf drei Magnetbandrollen von gleicher Länge (Standardlänge 731,5 m) gespeichert ist und auf einen Magnetplattenspeicher des Typs IBM 3330 übernommen werden soll, dann taucht die Frage auf, wieviele Plattenstapel bereitgestellt werden müssen. Diese Frage ist unter der Voraussetzung, daß der gleiche Blockungsfaktor benutzt wird, zu beantworten!

5) Bezugnehmend auf die Aufgabe 1) ist der Kennsatz des Formats 1 für die IBM 3330 niederzuschreiben.

6) Wieviele Sätze von 160 Bytes Länge können auf einem Magnetband gespeichert werden, das 731,5 m lang ist? Es kann vorausgesetzt werden, daß das Magnetband keine schadhaften Stellen aufweist. Ermittle die Satzanzahl für folgende Blockungsfaktoren:
a) 2, b) 3, c) 4, d) 8, e) 16.
Vergleiche die gefundenen Ergebnisse mit der Anzahl der Sätze, die auf das gleiche Magnetband aufgezeichnet werden können, wenn keine Blockung vorgenommen wird. Es sei weiterhin vorausgesetzt, daß das Magnetband eine Zeichendichte von 1 600 Bytes pro Zoll zuläßt, d.h. rd. 630 Bytes/cm.

7) Löse das gleiche Problem wie bei Aufg. 6), jedoch für eine Zeichendichte von 6 250 bpi (6250 Bytes/Zoll entsprechen rd. 1588 Bytes/cm)!

8) Angenommen, es sind auf eine IBM 2314 insgesamt 8 000 000 Sätze geschrieben worden, wobei jeder Satz drei Bytes lang ist. Der Zylinder 0 ist für das Datenträgerinhaltsverzeichnis (VTOC) und für die Kennsätze reserviert worden. Die Zylinder 200 bis 203 sind für Ersatzspuren (Ausweichspuren) vorgesehen. Wie groß muß der kleinste Blockungsfaktor gewählt werden, damit alle Sätze auf einem Plattenstapel Platz finden?

9) Es liege die gleiche Aufgabenstellung wie bei Aufg. 8) vor, nur sind die Verhältnisse für 800 000 Sätze zu untersuchen.

10) Welcher kleinste Blockungsfaktor ist zu wählen, wenn auf einem Plattenstapel der IBM 2314 insgesamt 800 000 Sätze von 50 Bytes Länge untergebracht werden sollen? Die Zylinder 0 sowie 200 bis 203 sind für die gleichen Zwecke reserviert, die in Aufg. 8) genannt sind.

10. Betriebssysteme

10.1. Programmiersprachen

Nachdem ein Programmierer zunächst eine Aufgabe erfaßt und umrissen und danach die Programmablaufpläne gezeichnet hat, wird er die erarbeitete Lösung in der englischen Sprache, mit mathematischen und logischen Symbolen durchsetzt, auszudrücken versuchen. Irgendwie muß dann freilich das Programm in die Maschinensprache der IBM /370 übersetzt werden, damit es auch ausgeführt werden kann. Eine solche Übersetzung kann grundsätzlich der Programmierer selbst durchführen, aber bei längeren und komplizierten Programmen sind sicher eine Menge Fehler zu erwarten und letzten Endes wird außerdem viel Zeit vergeudet. Die Übersetzungsarbeit ist grundsätzlich eine simple geisttötende Schreibarbeit. Jeder, der die Regeln der Maschinensprache beherrscht und der der englischen Sprache sowie der mathematischen Formelsprache, in der das Programm formuliert ist, mächtig ist, kann eigentlich die Übersetzung durchführen, freilich oft unter erheblichen Anstrengungen und Mühen. Das bei solchen Übersetzungen auftauchende Problem ist eine reine Zeit- und Kostenfrage: Es wird mehr als verantwortet werden kann Zeit verschwendet und außerdem muß man sich der Dienste eines relativ langsamen und zudem auch teuren Personals bedienen.

Computer sind in der Lage, derartige stupide Arbeiten viel akkurater und schneller zu erledigen. Deshalb sollte es auch möglich sein, ein Computerprogramm zu schreiben, das Instruktionen liest, die in einer bestimmten Sprache abgefaßt sind, und diese in eine andere Sprache übersetzt. Ein solches Programm ist freilich recht kompliziert und wird mehr Instruktionen umfassen als in einen Realspeicher hineinpassen. Der Grad der Komplexität eines Übersetzungsprogrammes ist sicher proportional der Komplexität der Sprache, die der Computer übersetzen soll.

Die einfachste Programmiersprache ist eine symbolische Form der Maschinensprache. Beispielsweise kann der Programmierer die Instruktionsfolge

```
AR      2,3
ST      2,XYZ
```

niederschreiben, wobei diese an die Stelle der beiden Instruktionen

```
1A 23
5Ø 2Ø B Ø67
```

der Maschinensprache tritt.

In diesem Fall ist der Übersetzungsprozeß beschränkt auf folgende Vorgänge:

a) Übersetzung der mnemonischen Operationsschlüssel in die Operationsschlüssel der Maschinensprache der IBM /370.

b) Aufbau einer Tabelle der symbolisch benannten Speicherplätze, wie in unserem Beispiel etwa

   XYZ ≙ Ø67 + (11),

   und nachfolgender Substitution dieser Symbole in den Instruktionen durch ihre Äquivalente von echten Maschinenadressen. Das Assemblierer-Programm [1] der IBM /370 nimmt als Eingabe symbolische Statements der oben aufgeführten Form

1) Im angloamerikanischen Sprachraum als Assembler bezeichnet.

auf und gibt als Ausgabe die entsprechenden Instruktionen in der Maschinensprache aus. Man sagt, daß Programme, die in dieser symbolischen Schreibweise abgefaßt sind, in der Assemblierersprache [1] notiert sind.

Im Jahre 1954 wurde eine künstliche Sprache geschaffen, die FORTRAN genannt wurde. Sie stellt ein Gemisch von englischen Wörtern und mathematischen Formelzeichen dar. Für sie entstand wenig später (1956) ein Übersetzungsprogramm, das in der Lage war, in der FORTRAN-Sprache geschriebene Statements in Instruktionen der Maschinensprache der IBM 704 zu übersetzen. Nachfolgend entstanden ähnliche Übersetzungsprogramme für beinahe alle gängigen Computertypen. In der FORTRAN-Sprache könnte der Programmierer etwa das folgende Statement niederschreiben:

```
C = XYZ + 67.89
```

Das Übersetzungsprogramm, als Kompilierer [2] bekannt, würde dieses eine Statement in eine Folge von Maschineninstruktionen übersetzen, die der Reihe nach den Wert von XYZ in ein Register laden, auf den Inhalt dieses Registers dann die im Hauptspeicher liegende Konstante 67.89 addieren und schließlich die so entstandene Summe auf einen Speicherplatz weglegen, der symbolisch unter dem Namen C bekannt ist. FORTRAN ist eine weit verbreitete Programmiersprache vor allem bei mathematisch-technischen Anwendungen, d.h. in der Technik und in den Ingenieurwissenschaften.

Im Jahre 1961 schuf ein vom Verteidigungsministerium der USA gegründetes Komitee eine Programmiersprache speziell für kommerzielle Probleme, die unter dem Namen COBOL bekannt wurde. Das dem FORTRAN-Statement

```
C = XYZ + 67.89
```

äquivalente Statement in COBOL lautet:

```
ADD 67.89 TO XYZ GIVING C.
```

FORTRAN wird immer dann vorgezogen, wenn ein Programm verwickelte Berechnungen einschließt. COBOL hingegen findet sein Einsatzgebiet vorzugsweise dort, wo die Eingabe/Ausgabe komplizierte Aufbereitungen erfordert. Beinahe Zweidrittel aller großen Computerinstallationen in der gesamten Welt benutzen bis zu einem gewissen Grade COBOL.

Eine gemeinsame Arbeitsgruppe von Benutzern großer IBM-Computer und von Programmexperten der IBM entwickelte im Jahre 1964 eine neue Programmiersprache, die PL/1 [3] genannt wurde. In PL/1 wurden viele wertvolle Eigenschaften von FORTRAN und COBOL vereinigt. Hinzu kamen weitere zusätzliche Verarbeitungsmöglichkeiten. Um bei einem Beispiel zu bleiben, sei nur die Möglichkeit der bequemen Handhabung verwickelter Zeichenketten erwähnt.

Zu der Zeit, in der die Computer begannen, die für Abrechnungszwecke eingerichteten Lochkartenausrüstungen abzulösen, war es die natürlichste Sache in der Datenverarbeitung, Programme zu schreiben, die diese Umstellung erleichterten. Unter all den hierbei entwickelten Verfahren zeichnete sich besonders ein Programm aus, das als RPG [4] bezeichnet wurde. Als Eingabe für dieses Programm dienen detaillierte Angaben über die Form der Eingabe- und Ausgabedaten und über die erforderlichen Be-

---

1) Im angloamerikanischen Sprachraum als Assembler Language, abgekürzt AL, bezeichnet. Die Abkürzung AL werden wir auch im Deutschen benutzen, da sie sich eingebürgert hat.

2) Im angloamerikanischen Sprachraum als Compiler bezeichnet.

3) PL/1 ist die Abkürzung von Programming Language / One

4) RPG ist die Abkürzung von Report Program Generator

rechnungen. Aus diesen Angaben wird ein Programm erzeugt, u. U. ein nicht gerade sehr effektives, das in der Lage ist, die vorgesehene Arbeit zu verrichten. Aus dem RPG hervorgegangene Programme sind für kleinere Computer sicher sehr nützlich, außerdem eignet sich der RPG besonders für kleinere Routinen, die zur Anfertigung von Geschäftsberichten benötigt werden. Diese Feststellungen gelten uneingeschränkt.

Eine sechste Programmiersprache, APL [1] mit Namen, kam auf, als Datenendstationen (Terminals) mit großen Computersystemen verbunden wurden. APL befähigt letzten Endes den Bediener eines Terminals, den Computer wie einen gigantischen Rechenschieber und Tischrechner zu benutzen.

Wie die Umgangssprachen unterliegen auch die Programmiersprachen einer ständigen Entwicklung und Änderung. Es nimmt deshalb nicht wunder, wenn bei Erscheinen neuer Übersetzer bzw. Übersetzerversionen gleichzeitig auch verschiedene Sprachänderungen bzw. -verbesserungen mitgeteilt werden. In den restlichen Kapiteln dieses Buches werden wir uns auf vier spezielle Programmiersprachen beschränken. Unsere Betrachtungen über diese Sprachen wollen wir dabei auf bestimmte Sprachübersetzer (Assemblierer bzw. Kompilierer) ausrichten, die häufig in der Praxis eingesetzt sind. Im einzelnen handelt es sich um folgende Sprachen und Übersetzer:

- PL/1 ⟶ Optimizing Compiler
- FORTRAN ⟶ F- bzw. G-Compiler
- COBOL ⟶ ANS-Compiler
- AL ⟶ Assembler/VS

Unter bestimmten Voraussetzungen (Normung) werden wir es uns jedoch vorbehalten, gesonderte Hinweise zu geben, was wir besonders bei FORTRAN für sinnvoll erachten.

Wir werden die Grundlagen aller vier Sprachen behandeln, unter keinen Umständen jedoch den Ehrgeiz entwickeln, irgendeine Sprache in voller Breite und Tiefe abzuhandeln. Nach dem Studium dieses Buches sollte der Leser in der Lage sein, sich der IBM-Handbücher bedienen zu können, um in die letzten Geheimnisse derjenigen Sprache einzudringen, über die er nähere Informationen einholen muß oder will.

10.2. Betriebssysteme

Unter einem Betriebssystem versteht man eine Menge von Programmen, die den kontinuierlichen Betrieb eines Computers, d.h. den Übergang von einem Benutzerprogramm zu einem anderen, sicherstellen, ohne daß ein Bedienereingriff erfolgen muß. Die IBM unterstützt drei Betriebssysteme, die auf die virtuelle Speichertechnik ausgerichtet sind. Jedes System stellt dabei eine natürliche Weiterentwicklung eines bewährten Betriebssystemes der IBM /360 dar. Das DOS/VS [2] ist in erster Linie für Installationen bestimmt, die mit einem realen Hauptspeicher von 256 k und weniger Bytes ausgerüstet sind. Das OS/VS1 [3] ist für DV-Systeme bestimmt, deren realer Hauptspeicher zwischen 200 k und 1024 k groß ist. Das OS/VS2 [3] schließlich ist für Installationen gedacht, deren Hauptspeicher mindestens 768 k Bytes aufweist. Da das OS/VS1 und das OS/VS2 sich durch viele gemeinsame Eigenschaften auszeichnen, wer-

1) APL ist die Abkürzung von A Programming Language

2) DOS/VS ist die Abkürzung von Disk Operating System/Virtual Storage, d.h. ein Plattenbetriebsystem für die virtuelle Speichertechnik

3) OS/VS ist die Abkürzung von Operating System/Virtual Storage, die nachfolgenden Ziffern 1 bzw. 2 geben die Ausbaustufen an.

den wir stets nur den Begriff OS benutzen, wenn wir uns auf beide beziehen wollen. Der Bequemlichkeit wegen wollen wir an Stelle von DOS/VS einfach nur vom DOS sprechen.

Trotz ihrer Unterschiede besteht jedes Betriebssystem aus vier grundsätzlichen Programmtypen:

1. *Anwendungsprogramme*
   Unter den Anwendungsprogrammen versteht man solche Programme, die auf die spezielle Installation ausgerichtet sind, d.h. deren Anwendungen abdecken.

2. *Sprachübersetzer*
   Hierzu zählen die Assemblierer und die Kompilierer für die Programmiersprachen wie AL, FORTRAN, COBOL, PL/1 usw.

3. *Dienstprogramme*
   Hierunter versteht man eine vom Hersteller zur Verfügung gestellte Sammlung von Programmen, die zur Ausführung der immer wiederkehrenden Arbeiten, wie Sortieren, Herstellung von Magnetbandkopien usw., vorgesehen sind.

4. *Steuerprogramme*
   Die Steuerprogramme stellen gewissermaßen das Herz eines jeden Betriebssystemes dar. Sie dienen zur Behandlung von Unterbrechungen, sie ermöglichen den Übergang von einem Programm zu einem anderen, sie informieren das Bedienungspersonal über anormale Geschehnisse usw.

Da die Bedürfnisse der einzelnen Installationen unterschiedlich sind, kann jedes Betriebssystem auf die speziellen Erfordernisse einer Installation zugeschneidert werden. Alle Programme sind auf SEDZ [1] untergebracht. Die SEDZ, auf der die Steuerprogramme gespeichert sind, heißt Systemresidenz . Insgesamt gesehen bilden alle Programme eines Betriebssystems die sogenannte Bibliothek der betreffenden Installation.

Herrührend von den Unterschieden zwischen den beiden Betriebssystemen DOS und OS weichen die für das DOS bereitstehenden Kompilierer ein wenig von denen ab, die für das OS geschaffen wurden. Beide OS-Betriebssysteme benutzen dagegen die gleichen Kompilierer.

10.3. Steuerprogramme

Das Subsystem der Steuerprogramme eines Betriebssystemes umfaßt im einzelnen die folgenden Programme:

1. *IPL-Lader* [2]
   Beim einleitenden Programmladen wird von der Hardware dieses Programm in den Hauptspeicher gelesen. Sinn und Zweck dieses Programmes besteht darin, unbenutzte Hauptspeicherplätze auf Null zu löschen, den Uhrzeitgeber zu prüfen oder zurückzusetzen usw. Allgemein gesprochen: Es werden bestimmte Verwaltungsaufgaben erledigt und zum Schluß wird der Supervisor eingelesen.

2. *Supervisor*
   Nach dem einleitenden Programmladen nimmt der Supervisor ständig die Hauptspeicherplätze mit den unteren Adressen ein. Bis zu 32 k werden von ihm auf die

---

1) SEDZ ist die Abkürzung von Speicher einheiten mit direkter Zugriffsmöglichkeit
2) IPL ist die Abkürzung von Initial Program Loading (auf deutsch: einleitendes Programmladen)

Dauer belegt. Sein genauer Umfang hängt davon ab, welches Betriebssystem ausgewählt wurde, und außerdem davon, um welche Optionen das Betriebssystem in der betreffenden Installation ergänzt wurde. Einige Teile des Supervisors werden im Realspeicher festgehalten, andere Teile hingegen werden virtuell gefahren. Der im virtuellen Speicher liegende Teil schließt z.B. Routinen ein, die die standardisierte Fehlerbeseitigung beinhalten, im Grunde genommen also Verfahren, die selten benötigt werden. Der Supervisor umfaßt eine standardisierte Bearbeitung aller möglichen Unterbrechungen, behandelt den Seitenwechsel und setzt die EA-Operationen auf. Er kann darüberhinaus auch EA-Anforderungen in Warteschlangen stellen und die erforderliche EA-Operation so bald als möglich starten, d.h. wenn das benötigte EA-Gerät wieder verfügbar ist.

3. *Job- oder Auftragssteuerung*
Nachdem ein Job oder Auftrag zu Ende gegangen ist, übergibt der Benutzer normalerweise die Programmablaufsteuerung zurück an den Supervisor, was er durch entsprechende Programmierung seines Anwendungsprogrammes erreicht. Der Supervisor selbst leitet dann über zum Jobsteuerungsprogramm. Dieses wiederum verarbeitet solange Abbildungen von Lochkarten, bis es einen speziell präparierten Satz findet, der ihm ansagt, daß ein neuer Job (Auftrag) abzuwickeln ist. Das Jobsteuerprogramm verarbeitet nun die Steuersätze (Steuerkarten) für diesen Job, sucht das zugehörige Programm auf, lädt es in den Hauptspeicher und übergibt an dieses schließlich die Programmablaufsteuerung. Dieses soeben beschriebene System von aufeinanderfolgenden Schritten eliminiert die Notwendigkeit von Eingriffen des Bedienungspersonals zwischen zwei Jobs.

4. *Datenmanagement*
Hierzu zählt eine Menge vieler Unterprogramme, die die gewöhnlichen und regelmäßigen EA-Aufgaben handhaben.

10.4. Dienstprogramme

Die bei den DV-Installationen vorhandenen Dienstprogramme schließen mit hoher Wahrscheinlichkeit ein Sortier- und Mischprogramm ein; mit diesem können die Sätze einer Datenmenge in eine vorgegebene Reihenfolge gebracht werden, oder es können die Sätze mehrerer Datenmengen zusammengemischt werden.

Die ebenfalls zu den Dienstprogrammen zählenden Hilfsprogramme umfassen Programme zum Initialisieren von Magnetplattenstapeln, zum Kopieren von Magnetplatten auf Magnetbänder und umgekehrt, zum Drucken des Inhalts einer auf Magnetplatten oder -bändern liegenden Datenmenge, zum Führen und Warten der Bibliotheken der betreffenden Installation usw.

Ein in jeder Installation vorhandenes Dienstprogramm ist der Binder , auch unter dem Namen Linkage Editor bekannt. Im ersten Moment scheint es wünschenswert zu sein, wenn die Ausgabe der Sprachübersetzer Maschineninstruktionen sind, die sich zum unmittelbaren Laden in den Hauptspeicher eignen und dadurch eine sofortige Ausführung ermöglichen. Obwohl Sprachübersetzer so geschrieben werden können, daß sie ein solches Vorgehen gestatten, würden sie wenig Sinn haben. Die meisten Programme sind so umfangreich, daß sie nicht auf einmal abgefaßt werden. Jeder Teil wird zweckmäßigerweise getrennt geschrieben, getrennt getestet und danach erst zum umfassenden Programm zusammengefügt. Das hierzu erforderliche Aufbereiten der einzelnen Teile für die abschließende Kombination und das Verbinden dieser Teile ist Aufgabe des Binders, des Linkage Editors.

Die Sprachübersetzer erzeugen Programmsteuerabschnitte. [1] Unter einem Programmsteuerabschnitt wollen wir eine Menge von Instruktionen und/oder Konstanten und Arbeitsgebiete verstehen, die nicht voneinander losgelöst werden können; er ist also der kleinste unteilbare Teil eines Programmes. Der Binder fügt die Steuerabschnitte so zusammen, daß sie ein Programm bilden. Wir werden im nächsten Abschnitt sehen, wie das geschieht.

Den Programmsteuerabschnitten ähnlich sind die sogenannten gemeinsamen Abschnitte. Diese enthalten nur Arbeitsgebiete, also keine Instruktionen und auch keine Konstanten. Der Binder behandelt sie fast genau so wie die Steuerabschnitte.

10.5. Objektmoduln

Die einem Sprachübersetzer zugeführte Eingabedatenmenge wird als Quellenmodul bezeichnet, die von ihm vorgenommene Ausgabe heißt Objektmodul. Jeder Objektmodul besteht aus einen oder mehreren Steuerabschnitten und gemeinsamen Abschnitten. Objektmoduln werden in Form von Kartenabbildern zusammengestellt, obgleich sie gewöhnlich auf Magnetplatten aufgezeichnet werden. Die folgenden vier verschiedenen Kartenabbilder (Kartenarten) werden von den Sprachübersetzern erzeugt [2]:

ESD	*Externes Symbolverzeichnis*
TXT	*Text*
RLD	*Verschiebungsverzeichnis*
END	*Modulende*

Betrachten wir den Inhalt und die Bedeutung der einzelnen Kartenarten der Reihe nach: Die ESD-Karten enthalten den Namen jedes Abschnittes sowie den Namen jedes Symbols, das im Quellenmodul als Eingangsname definiert ist. Unter einem Eingangsnamen wird jeder einer Instruktion oder einem Datenelement gegebener Name verstanden, der es dem Programmierer gestattet, sich auf ihn in anderen Objektmoduln zu beziehen. Solche Namen nützen außerordentlich viel, wenn es darum geht, Teile eines umfangreichen Programmes miteinander zu verbinden. Die TXT-Karten enthalten die aktuellen Maschineninstruktionen und die Konstanten. RLD-Karten werden für die sogenannten Adreßkonstanten oder kurz Adkons produziert. Darunter verstehen wir Konstanten, die Adressen von Datenelementen oder von Instruktionen enthalten. Die END-Karte markiert das Ende eines Objektmoduls. Sie kann darüberhinaus noch den Abstand [3] zwischen dem Anfang des Moduls und seiner ersten auszuführenden Instruktion beinhalten. Unter der ersten auszuführenden Instruktion soll diejenige verstanden werden, die als erste zum Zuge kommt, wenn im Verlauf der Programmausführung dem betreffenden Modul die Programmablaufsteuerung übertragen wird.

Der Binder oder Linkage Editor nimmt als Eingabe Objektmoduln sowie in manchen Fällen zusätzliche Steuerkarten an. Die von ihm erzeugte Ausgabe wird als Lademodul bezeichnet. Beim Lademodul sind alle Programmabschnitte und gemeinsame Abschnitte der Objektmoduln zu einem einzigen Modul verbunden. Der Binder versucht

1) Im angloamerikanischen Sprachraum als Control Sections oder kurz CSECT's bezeichnet (deutsch auch kurz: Programmabschnitt)

2) Die Abkürzungen stammen von folgenden Begriffen her:
ESD — External Symbol Dictionary
TXT — Text
RLD — Relocation Dictionary
END — End of Modul

3) Im angloamerikanischen Sprachraum als „Offset" bezeichnet.

ebenfalls die Adreßkonstanten aufzulösen. Das geschieht in folgender Weise: Zunächst wird versucht, den Wert der Adreßkonstanten innerhalb des Objektmoduls zu bestimmen, in dem sie auftaucht. Wenn das nicht möglich ist, so wird das externe Symbolverzeichnis (aus den ESD-Karten aufgebaut) nach Eintragungen durchsucht, die den entsprechenden Eingangsnamen oder Abschnittsnamen aufweisen. Bleibt ein solcher Suchvorgang ebenfalls ohne Erfolg, so werden eine oder mehrere Bibliotheken durchforscht, die gewöhnlich auf dem Systemplattenstapel residieren; gesucht wird dabei nach einem Modul, der den zu findenden Namen beinhaltet. Solche Moduln werden in das Programm eingebracht, gerade so als ob der Programmierer sie selbst eingeschlossen hätte. Einen solchen Prozeß bezeichnet man als automatisches Binden [1].

Wenn es unmöglich ist, eine Adreßkonstante aufzulösen, so veranlaßt der Binder das Drucken einer Mitteilung und setzt Nullen auf den betreffenden Speicherplatz. Ein Programm mit solchen unaufgelösten Adreßkonstanten wird freilich nur dann korrekt ablaufen, wenn es auf keine Bezugnahmen zu derartigen Adressen aufläuft.

Die für das System IBM /370 geschaffenen Binder lösen die Adreßkonstanten nur soweit auf, daß sie den Abstand vom Beginn des Lademoduls feststellen. Dadurch, daß die absoluten Adressen erst beim tatsächlichen Laden des Programmes festgesetzt und verbunden werden, kann der Lademodul in jedes Hauptspeichergebiet verschoben werden. Der Supervisor erledigt beim endgültigen Laden des Programmes die letzte entscheidende Verschiebung.

10.6. Die logischen EA-Einheiten des Betriebssystems DOS/VS

In diesem und in den folgenden fünf Abschnitten behandeln wir soviele Einzelheiten der Jobsteuerung (Auftragssteuerung), daß der Leser damit in die Lage versetzt ist, Programme, die unter dem Betriebssystem DOS/VS laufen sollen, zu übersetzen, zu binden und auszuführen.

Weil mehrere EA-Geräte der gleichen Art an eine IBM /370 angeschlossen werden können, wurde für das Betriebssystem eine äußerst zweckdienliche Funktion geschaffen, die sich darin zeigt, daß das Maschinenbedienungspersonal ein bestimmtes Gerät durch ein ähnliches ersetzen kann, d.h. z.B. daß ein Drucker durch einen anderen ersetzt werden kann. Für den Programmierer hat diese Eigenschaft zur Folge, daß er in seinem Programm nur den Gerätetyp und einen symbolischen Namen für jedes Gerät aufzuführen braucht. Dann liegt es in der Verantwortlichkeit des Operateurs, den symbolischen Namen in irgendeiner Weise dem gerade einzusetzenden Gerät zuzuweisen.

Wenn ein Programmierer diese Möglichkeit des Betriebssystemes ausnutzt, so obliegt es ihm, sich in seinem Programm symbolischer Namen der Form SYSxxx oder SYSnnn zu bedienen. Hierbei bedeutet xxx eine Folge von drei Buchstaben und nnn eine dreiziffrige Zahl im Wertebereich zwischen 000 und 241. Jeder solche Name bezieht sich dabei auf eine logische EA-Einheit. Symbolische Namen der Form SYSnnn kennzeichnen dabei speziell die logischen Einheiten des Programmierers [2], Namen der Form SYSxxx logische Einheiten des Systems.

Für xxx kann eine der Buchstabenfolgen RDR, IPT, PCH, LST, LOG, LNK, RES, SLB, RLB, REC, CLB, VIS oder CAT eingesetzt werden. Den somit benannten logischen Systemeinheiten kommen die folgenden speziellen Funktionen zu:

---

1) Im angloamerikanischen Sprachraum als „Autolink" bezeichnet.

2) Kurz auch logische Programmierereinheiten genannt

a) SYSRDR
Dieser Name bezieht sich auf den Systemleser. Gewöhnlich verbirgt sich dahinter ein Lochkartenlesegerät. An seiner Stelle können aber durchaus auch Magnetband- oder Magnetplattenspeichergeräte eingesetzt werden. Über SYSRDR werden die Steuersätze für das Betriebssystem gelesen.

b) SYSIPT
Dieser Name bezieht sich auf das Standardsystemeingabegerät. Gewöhnlich stellt dieses ein Lochkartenlesegerät dar, an seine Stelle kann jedoch durchaus auch ein Magnetband- oder Magnetplattenspeichergerät treten. Der Name SYSIPT kann grundsätzlich dem gleichen EA-Gerät zugeordnet werden, dem der Name SYSRDR gegeben wurde.

c) SYSPCH
Dieser Name bezieht sich auf das Systemstanzgerät. Gewöhnlich wird hierfür ein Lochkartenstanzer bereitgestellt, an dessen Stelle jedoch auch ein Magnetband- oder Magnetplattenspeichergerät treten kann.

d) SYSLST
Dieser Name bezieht sich auf das Standardsystemausgabegerät. Gewöhnlich wird hierfür ein Zeilendrucker bereitgestellt, an dessen Stelle jedoch auch ein Magnetband- oder Magnetplattenspeichergerät treten kann. Wird ein Magnetbandgerät benutzt, so kann es grundsätzlich identisch mit demjenigen sein, dem der Name SYSPCH gegeben wurde.

e) SYSLOG
Dieser Name bezieht sich auf die Konsole, über die Nachrichten an den Operateur ausgegeben werden und über die er Mitteilungen an das System, d.h. z.B. vom Betriebssystem erwartete Antworten, eingeben kann. Der Name SYSLOG kann auch einem Zeilendrucker gegeben werden, doch ist in einem solchen Fall natürlich kein Bedienereingriff mehr möglich.

f) SYSLNK
Dieser Name bezieht sich stets auf das Magnetplattenspeichergerät, das als Eingabegerät für den Binder (Linkage Editor) eingesetzt wird.

g) SYSRES
Auf dem mit diesem Namen versehenen Gerät residiert das Betriebssystem.

h) SYSSLB
Dieser Name bezieht sich auf ein Gerät, auf dem eine private Bibliothek mit Quellenstatements (Quellenmoduln) untergebracht ist, die an Stelle der vom System unterstützten gleichartigen Bibliothek benutzt wird.

i) SYSRLB
Dieser Name bezieht sich auf ein Gerät, auf dem eine private Bibliothek mit verschieblichen Statements (Objektmoduln) untergebracht ist, die an Stelle der vom System unterstützten gleichartigen Bibliothek benutzt wird.

j) SYSREC
Dieser Name bezieht sich auf ein Gerät, auf dem die Fehlersätze der Geräteanzeigeleisten (Logouts) festgehalten werden.

k) SYSCLB
Dieser Name bezieht sich auf ein Gerät, auf dem eine private Bibliothek ausführbarer Programme (Lademoduln) untergebracht ist, die an Stelle der vom System unterstützten gleichartigen Bibliothek benutzt wird.

l) SYSVIS
Dieser Name bezieht sich auf ein Gerät, das als virtueller Speicher für das System eingesetzt ist.

m) SYSCAT
Dieser Name bezieht sich auf ein Gerät, das den Katalog für VSAM enthält, wenn VSAM Teil des Systems ist.

Wenn die symbolischen Einheitennamen SYSRDR und SYSIPT dem gleichen Gerät zugewiesen sind, so kann sich der Operateur, jedoch nicht der Programmierer, auf beide logischen Systemeinheiten unter dem Sammelnamen SYSIN beziehen. SYSIN muß sogar benutzt werden, wenn beide logischen Einheiten durch den gleichen Bereich auf einem Plattenspeichergerät realisiert sind. Wenn die symbolischen Einheitennamen SYSPCH und SYSLST dem gleichen Magnetbandgerät zugewiesen sind, so muß sich der Operateur des Namens SYSOUT bedienen, damit er sich auf beide logischen Einheiten beziehen kann.

Im allgemeinen sind die logischen Einheiten des Programmierers für Benutzerzwecke reserviert, doch verwenden die Kompilierer die Namen SYSØØ1, SYSØØ2, ···, SYSØØ5 für Arbeitsdateien.

Wenn das Betriebssystem eingerichtet oder generiert wird, so können einige oder alle logischen Systemeinheiten permanent gewissen EA-Geräten zugewiesen werden. Derartige Zuweisungen sind in der Regel auch vorgesehen, doch sind sie von Installation zu Installation unterschiedlich. Zusätzlich ist es möglich, den maximalen Wert von nnn für die logischen Einheiten SYSnnn des Programmierers zu limitieren, so daß z.B. SYSØØ9 oder SYSØ25 diejenigen symbolischen Namen sind, die gerade noch verwendet werden dürfen. Während des einleitenden Programmladens besteht eine der Funktionen des IPL-Programmes darin, die Adresse des SYSRES zugeordneten Gerätes in eine Tabelle zu stellen, die vom Supervisor verwaltet wird. Diese Zuweisung von SYSRES bleibt bis zum nächsten einleitenden Programmladen bestehen. Die Zuweisungen für SYSVIS und SYSCAT hingegen können während des einleitenden Programmladens geändert werden. Danach bleiben sie bis zum nächsten einleitenden Programmladen unverändert.

10.7. Gerätezuweisungen im Betriebssystem DOS/VS

Mitunter müssen die im System verankerten Standardzuweisungen bezüglich der EA-Geräte überschrieben werden. Hierzu kann entweder das Jobsteuerstatement ASSGN oder der entsprechende Operateurbefehl ASSGN benutzt werden. Das ASSGN-Statement besitzt folgende allgemeine Form:

```
                                  [,X'C8']
                                  [,X'CØ']  [,TEMP]
                                  [,X'DØ']  [,PERM] [,VOL=vvvvvv [,SHR]]
//bASSGNbSYSxxx,{ziel}            [,ALT]
               {IGN}              [,H1]
               {UA }              [,H2]
```

Die geschweiften Klammern zeigen an, daß eine Wahl zwischen den darin verzeichneten Möglichkeiten getroffen werden muß. Die eckigen Klammern hingegen bedeuten grundsätzlich das gleiche, doch wird durch sie zusätzlich angezeigt, daß der gesamte Klammerninhalt auch einfach weggelassen werden kann. Dieser Standardschreibweise wollen wir uns im gesamten Buch bedienen.

Durch b wird angesagt, daß an der bezeichneten Stelle eine oder mehrere Leerzeichen stehen können. Wenn das Statement auf eine Lochkarte abgelocht wird, so müssen die beiden Schrägstriche auf den Spalten 1 und 2 abgelocht werden. Das gesamte Statement muß außerdem vor der Spalte 72 enden. Leerzeichen sind nur vor und nach dem Schlüsselwort ASSGN erlaubt, sonst nirgends. Bemerkungen oder Kommentare können zusätzlich auf der Lochkarte untergebracht werden. Wenn sie benutzt werden sollen, so müssen sie, vom Statement durch ein oder mehrere Leerzeichen getrennt, auf den letzten Operanden folgen. Ein um eine Bemerkung erweitertes typisches Steuerstatement lautet z.B. wie folgt:

```
// ASSGN SYSØ16,X'ØØE'  BEISPIELSKOMMENTAR
```

Dieses Statement besagt, daß die logische EA-Einheit SYSØ16 mit dem Gerät verknüpft wird, das am Kanal Ø mit der Adresse ØE angeschlossen ist.

Unter xxx ist entweder eine der für Systemeinheiten vorgesehenen Buchstabenfolgen (siehe Abschnitt 10.6.) oder eine der für die betreffende Installation zulässigen Einheitennummern für die logischen Programmiereinheiten einzusetzen.

Doch wenden wir uns nun der Bedeutung des Inhalts der geschweiften Klammer zu. Bei Niederschrift von IGN werden Ausgabeanweisungen im Benutzerprogramm vom Supervisor als erfolgreich ausgeführt vermerkt, ohne daß jedoch die Operation tatsächlich stattgefunden hat. Für die logischen Einheiten SYSRDR, SYSIPT, SYSIN und SYSCLB darf IGN nicht angegeben werden. Bei der Eingabe hängt die getroffene Maßnahme von der verwendeten Programmiersprache ab; bei den meisten Sprachen bewirkt die erste Eingabeanweisung, daß dem Benutzerprogramm das Auftreten einer Dateiendebedingung signalisiert wird. Durch UA wird festgelegt, daß die betreffende logische Einheit SYSxxx keinem EA-Gerät zugewiesen werden soll. Jede Bezugnahme auf eine solche logische Einheit hat den Abbruch des jeweiligen Programmes zur Folge. Unter ziel kann eine der folgenden Aussagen stehen:

a) X'kgg'
Durch kgg wird das der logischen Einheit SYSxxx zuzuweisende EA-Gerät angezeigt, wobei k den Kanal und gg die Adresse des EA-Gerätes bedeutet. Statt X'kgg' kann auch die Listenform

(X'kgg'[X'kgg'] ···)

benutzt werden, wodurch eine Zuweisung von bis zu sieben EA-Geräten zu einer logischen Einheit möglich wird. Die Liste wird hierbei von links nach rechts durchsucht: Auf das erste freie unbeschäftigte Gerät wird schließlich die tatsächliche Zuweisung bezogen.

b) SYSyyy
Bei der Angabe von SYSyyy erfolgt die Zuweisung von SYSxxx zum gleichen Gerät, das gegenwärtig der logischen Einheit SYSyyy zugewiesen ist.

c) READER
Die logische Einheit SYSxxx wird dem ersten Lochkartenleser zugewiesen, der vom System gerade als frei und unbeschäftigt eingestuft ist. Die Typbezeichnung der hierfür möglichen Geräte lautet: 25Ø1, 254ØR (Leseseite der IBM 2540), 35Ø4, 35Ø5, 1442N1, 2596, 252ØB1, 3525RP, 256Ø, 5425.

d) PRINTER
Die logische Einheit SYSxxx wird dem ersten freien Zeilendrucker zugewiesen. Die Typbezeichnungen der hierfür möglichen Geräte lauten: 14Ø3, 1443, 14Ø3U (mit dem Universaltypensatz), 32Ø3, 3211, 52Ø3, 52Ø3U.

e) PUNCH
Die logische Einheit SYSxxx wird dem ersten freien Lochkartenstanzer zugewiesen. Die Typbezeichnungen der hierfür möglichen Geräte lauten: 252ØB2, 252ØB3, 254ØP (Stanzerseite der IBM 2540), 1442N2, 3525P, 2596, 1442N1, 252ØB1, 3525RP, 256Ø, 5425.

f) TAPE
Die logische Einheit SYSxxx wird dem ersten freien Magnetbandgerät zugewiesen. Die Typbezeichnungen der hierfür möglichen Geräte lauten: 24ØØT9, 341ØT9, 342ØT9.

g) DISK
Die logische Einheit SYSxxx wird dem ersten freien Magnetplattenspeichergerät zugewiesen. Die Typbezeichnungen der hierfür möglichen Geräte lauten: 2311, 2314, 2319, 333Ø, 3333, 334Ø, 334ØR.

h) typ
Die logische Einheit SYSxxx wird einem Gerät mit der Typbezeichnung typ zugewiesen. Die zulässigen unter typ aufführbaren Typbezeichnungen sind unter READER, PRINTER, PUNCH, TAPE bzw. DISK vermerkt.

Bei Magnetbandgeräten, aber nur bei Magnetbandgeräten, kann zusätzlich einer der folgenden Operanden angegeben werden:

a) X'C8'
Aufzeichnungsdichte: 800 bpi

b) X'CØ'
Aufzeichnungsdichte: 1600 bpi

c) X'DØ'
Aufzeichnungsdichte: 6250 bpi

d) ALT
Durch ALT wird angezeigt, daß zusammen mit dem zuerst zugeordneten Magnetbandgerät ein zweites Magnetbandgerät (Wechselbandgerät) benutzt werden soll. Das Wechselbandgerät kommt dann zum Zuge, wenn das Rollenende beim ursprünglichen Bandgerät erreicht ist. Alle Angaben, die für die ursprüngliche Zuweisung gelten, werden für das Wechselbandgerät übernommen. Derartige Zuweisungen von Wechselbandgeräten erlauben die weitere Verarbeitung zu der Zeit, während der die erste Bandrolle zurückgespult und entladen wird.

Für die Lochkartengeräte IBM 2560 und IBM 5425 kann einer der folgenden beiden Operanden zusätzlich angegeben werden:

a) H1
Ablagefach 1

b) H2
Ablagefach 2

Wenn weder H1 noch H2 niedergeschrieben ist, wird H1 angenommen. H2 ist allein gültig bei den logischen Einheiten SYSRDR, SYSIPT oder SYSPCH.

Die beiden in der nächsten eckigen Klammer stehenden Zusatzoperanden haben folgende Bedeutung:

a) TEMP
Durch diese Angabe wird ausgesagt, daß die durch das betreffende Statement festgesetzte Zuweisung nur bis zum Ende des laufenden Jobs (Auftrags) andauern soll (temporäre Zuweisung).

b) PERM

Diese Angabe besagt, daß die Zuweisung solange Gültigkeit besitzen soll, bis entweder eine neue permanente Zuweisung, d.h. eine Zuweisung mit dem Operanden PERM, erfolgt oder ein neues einleitendes Programmladen (IPL) durchgeführt wird.

Wenn weder TEMP noch PERM niedergeschrieben wird, so gilt standardmäßig für das ASSGN-Steuerstatement die Angabe TEMP.

Durch den Operanden

VOL=vvvvvv

wird die laufende Nummer (Seriennummer) vvvvvv des Magnetbandes oder des Plattenstapels spezifiziert. Wenn der betreffende Datenträger auf irgendein freies durch den zweiten Operanden bezeichnetes Laufwerk aufgesetzt wird, so wird dieses Laufwerk auch zugewiesen. Im anderen Fall wählt das Betriebssystem das erste freie Laufwerk aus und schickt an den Operateur eine Nachricht hinaus, in der ihm mitgeteilt wird, auf welches Laufwerk der Datenträger montiert werden soll.

Der Operand SHR ist nur dann zulässig, wenn sich die Zuweisung auf ein Magnetplattenspeichergerät bezieht. Durch ihn wird die Möglichkeit geschaffen, daß ein Laufwerk gemeinsam von mehreren Jobs benutzt werden kann, d.h. von Jobs, die in mehreren Partitionen [1] (Speicherunterteilungen) gleichzeitig ablaufen. Wird dieser Operand weggelassen, so kann das betreffende Laufwerk nur von einem einzigen Job angesprochen werden.

Ein ASSGN-Operateurbefehl ähnelt dem ASSGN-Steuerstatement; die führenden Schrägstriche sind jedoch wegzulassen. Außerdem gilt als Standardoperand für die Zuweisungsdauer die Angabe PERM und nicht wie beim Steuerstatement die Angabe TEMP.

Die hinsichtlich SYSOUT und SYSCLB erfolgenden Zuweisungen müssen auf Dauer erfolgen. Zusätzlich dürfen logische Systemeinheiten nicht solchen Magnetbandgeräten zugewiesen werden, die gerade für eine andere logische Einheit gelten. Wenn also beispielsweise die logische Einheit SYSØ1Ø gerade dem Gerät mit der Adresse 180 zugewiesen ist, wird das Steuerstatement

```
// ASSGN SYSPCH,X'18Ø'
```

vom Betriebssystem abgelehnt, es sei denn, ihm geht ein Steuerstatement

```
// ASSGN SYSØ1Ø,X'181'
```

voraus oder ein diesem äquivalentes, etwa das Statement

```
// ASSGN SYSØ1Ø,UA
```

Im letzteren Fall darf SYSØ1Ø von dem betreffenden Job nicht benutzt werden.

Wenn im Mehrprogrammbetrieb [2] mehrere Jobs nebeneinander ausgeführt werden, darf mit Ausnahme von SYSLOG keine weitere logische Einheit mehr als einem Job zugewiesen werden, es sei denn, daß bei Zuweisungen von Magnetplattenspeichergeräten SHR spezifiziert ist.

---

1) Unter einer Partition versteht man den Teil eines Speichers, der für eine bestimmte Arbeit (Job, Auftrag) zur Verfügung steht. Entsprechend der Speichertechnik ist zwischen realen und virtuellen Partitionen zu unterscheiden.

2) Auch als Multiprogramming bezeichnet

Eine logische Einheit, die einem Magnetband- oder Magnetplattenspeichergerät zugewiesen ist, kann neu zugewiesen werden, wozu das folgende Steuerstatement zu benutzen ist:

```
//bCLOSEbSYSxxx [ ,X'kgg' [,X'ad'] ]
                [ ,IGN            ]
                [ ,UA             ]
                [ ,ALT            ]
```

Die Bedeutung der einzelnen Operanden ist bereits beim ASSGN-Steuerstatement erläutert worden. Ein Hinweis auf X'ss' möge dieses Thema abrunden: mit ad ist die Aufzeichnungsdichte bei Magnetbändern gemeint.

Bei Entdeckung eines solchen Statements werden auf Magnetbändern Bandmarken und das Datenträgerendetikett (EOV-Kennsatz) geschrieben; nachfolgend vollzieht sich das Rückspulen und Entladen.

Der Operateurbefehl CLOSE (keine Schrägstriche auf den ersten beiden Spalten) ist für SYSIN, SYSRDR, SYSIPT, SYSPCH bzw. SYSLST nur dann zulässig, wenn die betreffende Einheit einem Magnetplattenspeichergerät zugewiesen ist. Operateurbefehl und Steuerstatement CLOSE gelten für SYSPCH, SYSLST, SYSOUT und alle logischen Programmierereinheiten dann und nur dann, wenn die Einheit einem Magnetbandgerät zugewiesen ist.

Unterbleiben Neuzuweisungen, so behält eine Einheit ihre gegenwärtige Zuweisung.

Zum Schluß noch ein Hinweis zu den beiden Operateurbefehlen ASSGN und CLOSE: wegen des Fehlens der beiden Schrägstriche braucht vor den Schlüsselwörtern nicht unbedingt ein Leerzeichen oder mehrere Leerzeichen zu erscheinen.

10.8. Die Jobsteuerung (Auftragsteuerung) beim Betriebssystem DOS/VS

In diesem Abschnitt wollen wir kurz die Steuersätze oder Steuerstatements [1] umreißen, die erforderlich sind, um ein Programm zu übersetzen, seine Bestandteile zu binden und auszuführen. Nachdem wir ein Programm niedergeschrieben haben, besteht nur eine trügerische Hoffnung, daß es auf Anhieb korrekt abläuft. Infolgedessen werden wir es zunächst erst einmal nur übersetzen. Die während der Übersetzung gedruckten Fehlernachrichten helfen uns, den Quellenmodul zu bereinigen. Danach sollten wir sofort versuchen, die neue Übersetzung durch den Binderlauf und die Ausführung zu ergänzen. Die Steuersätze (Steuerkarten) zur Festlegung der Standardetiketten (Standardkennsätze) behandeln wir erst im nächsten Abschnitt, so daß hier noch eine gewisse Lücke verbleiben muß. Im Anhang K finden wir eine Übersicht sowohl über die Jobsteuerung als auch über das standardisierte Vorgehen beim einleitenden Programmladen.

Die Jobsteuerstatements besitzen die folgende Form:

```
// schlüsselwort operand [kommentar]
```

Die Spalten 1 und 2 nehmen stets zwei Schrägstriche (//) auf. Dem Schlüsselwort müssen ein oder mehrere Leerzeichen sowohl vorausgehen als auch nachfolgen. An

1) Die Steuersätze (Steuerstatements) sind auf Lochkartenformat ausgelegt. Da sie auch im allgemeinen auf Lochkarten abgelocht werden, spricht man meist einfach von Steuerkarten. Diesem Usus werden wir uns mitunter anschließen, vor allem dann, wenn wir unsere Erklärungen der Deutlichkeit wegen auf Lochkarten auslegen.

Stelle von operand sind ein Operand oder mehrere Operanden niederzuschreiben. Bei mehreren Operanden sind die einzelnen Operanden durch Kommas voneinander zu trennen. Innerhalb von operand dürfen keine Leerzeichen auftreten. Wird ein Kommentar für zweckmäßig erachtet, so muß er im Anschluß an operand folgen, jedoch sich von diesem Steuersatzteil durch ein oder mehrere Leerzeichen abheben. Im allgemeinen dürfen die Operanden nicht über die Spalte 71 hinausreichen und die Spalte 72 muß leer bleiben. Die Spalten 73 bis 80 können beliebig ausgefüllt werden. Die Jobsteuerstatements werden gewöhnlich über die logische Systemeinheit SYSRDR eingelesen, sie können aber auch über SYSLOG eingegeben werden.

Unter einem Job oder Auftrag versteht man eine Programmenge, die zur Ausführung durch den Computer vorgesehen und bereitgestellt wird. Die einzelnen Programme einer Menge sind in der Regel aufeinander bezogen. Der Jobname oder Auftragsname besteht aus 1 bis 8 Zeichen, wovon das erste ein Buchstabe sein muß. Die restlichen Zeichen können Buchstaben oder Ziffern sein. Die drei Zeichen $, # und @ werden in den Jobsteuerstatements als Buchstaben betrachtet. Der einzige Sinn des Jobnamens besteht darin, daß das Betriebssystem über einen Namen verfügt, unter dem es einen Job identifizieren kann. Z.B. kann es diesen Namen in Konsolnachrichten einrücken.

Jeder Job beginnt mit dem JOB-Steuerstatement. Es lautet allgemein

```
// JOB jobname
```

Das Ende eine Jobs wird durch das Jobendestatement angezeigt, das in der folgenden Form gebraucht werden kann:

```
/& [kommentar]
```

Durch die Zeichenkombination /& auf den ersten beiden Spalten wird das Jobende signalisiert. Wenn SYSRDR und SYSIPT auf getrennte Kartenlesegeräte bezogen sind, dann ist /& zur Kennzeichnung des Jobendes bei beiden Einheiten erforderlich.

Unter einem Jobstep oder Auftragsteil versteht man die Ausführung eines einzelnen Programmes zusammen mit seinen Unterprogrammen, d.h. mit denjenigen Routinen, die es aufrufen kann. Jeder Jobstep ist durch ein EXEC-Steuerstatement anzukündigen, das den folgenden Aufbau besitzt:

```
// EXEC  PGM=[programme]
```

Die Zeichenfolge PGM= kann, aber muß nicht benutzt werden; jedenfalls weist sie keinen Einfluß auf die Programmausführung auf. Unter programmname muß der Name des Programmes erscheinen, das ausgeführt werden soll. Unter dem Programmnamen versteht man denjenigen Namen, der einem Programm zugewiesen wurde, als es in die Bibliothek ausführbarer Programme (Bibliothek der Lademoduln) aufgenommen wurde. Die Namen der normalerweise eingesetzten Übersetzer (Übersetzernamen) heißen:

a) *PL/1-Übersetzer:* PLIOPT
b) *FORTRAN-Übersetzer:* FFORTRAN
c) *COBOL-Übersetzer:* FCOBOL
d) *AL-Übersetzer:* ASSEMBLY

Hinzu kommt LNKEDT als Name für den Binder (Bindername). Um ein Programm auszuführen, das gerade gebunden, jedoch noch nicht in die Bibliothek der Lademoduln aufgenommen und katalogisiert ist, genügt eine Reduzierung des EXEC-Statements auf die Form:

```
// EXEC
```

Der Operand fehlt also. Ein Programm, das dem Bindeprozeß unterlegen ist, wird nämlich erst dann katalogisiert, wenn das Jobsteuerstatement eingelesen ist.

Beim EXEC-Statement können u.U. auch zusätzliche Operanden erforderlich werden. So können wir z.B.

```
// EXEC [PGM=]programme,REAL
```

oder

```
// EXEC ,REAL
```

schreiben. Der hinzugefügte Operand REAL besagt, daß das Programm ausschließlich im realen Speicher auszuführen ist und nicht unter Zugrundelegung des virtuellen Speichers. Abgesehen von einigen Datenfernverarbeitungsprogrammen sollten einige wenige Programme immer im Realspeicher ausgeführt werden. In der zweiten o.a. Form zeigt das REAL vorangehende Komma die Auslassung des Programmnamens an.

Schreiben wir

```
// EXEC [PGM=]programme [,REAL],SIZE=umfang
```

oder

```
// EXEC [,REAL],SIZE=umfang
```

Durch diese Form des EXEC-Steuerstatements erfolgt eine Beschränkung des auszuführenden Programmes auf den angegebenen virtuellen Speicherumfang. Unter umfang können folgende Spezifikationen gemacht werden:

a) nK
   Hierbei ist unter n ein Vielfaches von 2 anzugeben, d.h. eine gerade ganze Zahl. Der durch nK festgelegte Speicherumfang ergibt sich dann aus dem n-fachen von 1024 Bytes.

b) AUTO
   Durch diese Angabe wird ein Speicherumfang für die Programmausführung festgelegt, der gleich dem ist, der zur Zeit des Bindens spezifiziert wurde.

c) (AUTO,nK)
   Für das auszuführende Programm wird ein Speicherumfang bereitgestellt, der gleich der Summe aus dem zur Bindezeit angegebenen Wert (AUTO) und einem zusätzlichen Betrag von nK Bytes, d.h. 1024·n Bytes, besteht.

Erscheinen sowohl REAL als auch SIZE als Operanden im EXEC-Statement, so gibt der bei SIZE erscheinende Wert den Umfang der realen Speicherpartition an, in der das Programm ausgeführt werden soll. Ist dagegen REAL weggelassen, so wird durch SIZE die Benutzerpartition im Hauptspeicher in zwei Teile zerlegt. Der erste Teil umfaßt die Speicherplätze vom Beginn der Partition an, und zwar soviele, wie aus der bei SIZE erscheinenden Angabe resultieren. Der zweite Teil umfaßt die darauffolgenden Speicherplätze bis zum Ende der Partition. Der erste Teil steht für die Programmausführung uneingeschränkt zur Verfügung, der zweite Teil wird vom Betriebssystem verwaltet. Eine spezielle durch GETVIS veranlaßte Operation kann dabei eingesetzt werden, um Stücke hiervon dem Programm zur Benutzung bereitzustellen. Fehlen sowohl REAL als auch SIZE, so gilt die gesamte Partition als zum ersten Teil zugeordnet, d.h. für GETVIS steht kein Speicherplatz mehr zur Verfügung.

Der Operand SIZE darf weder für die Übersetzer noch für den Binder [1] angegeben werden. Er muß unbedingt für solche Programme fixiert werden, die VSAM benutzen. Im allgemeinen wird wohl SIZE=AUTO festzulegen sein.

Dem EXEC-Steuerstatement müssen die erforderlichen ASSGN-Statements vorausgehen. Ebenso kann vor ihm noch ein OPTION-Statement erscheinen, das dann unumgänglich ist, wenn die eingerichteten Standardoptionen nicht ausreichen bzw. verfehlt sind. Das OPTION-Steuerstatement sieht wie folgt aus:

```
// OPTION option [,option]···
```

Die möglichen Optionen sind im Anhang K aufgeführt. Als Standardoptionen gelten diejenigen, die bei der Systemgenerierung ausgewählt und eingerichtet wurden.

Wenn ein auszuführendes Programm auf Magnetbänder bzw. Magnetplattenstapel zurückgreift, die durch Standardetiketten (Standardkennsätze) ausgezeichnet sind, so müssen dem EXEC-Statement zusätzlich noch Kennsatzsteuerstatements vorausgehen. Diese behandeln wir im nächsten Abschnitt 10.9.

Unmittelbar vor dem EXEC-Statement für den Binder[2] muß ein Steuerstatement von der Form

```
// LBLTYP TAPE
```

oder der Form

```
// LBLTYP NSD(ii)
```

erscheinen, wenn das zu bindende Programm Datenmengen anspricht, die der indexsequentiellen oder regionalen Organisationsform unterliegen, bzw. zu Magnetbändern mit Standardetiketten gehören. Werden ausschließlich nur Magnetbänder mit Standardkennsätzen eingesetzt, so ist das LBLTYP-Statement in der ersten Fassung heranzuziehen, d.h. mit dem Operanden LABEL. Im anderen Falle gibt ii in der Klammer des NSD-Operanden die Gesamtzahl der Bereiche derjenigen Datei an, die die größte Anzahl hiervon besitzt. Nicht zu zählen sind Dateien mit konsekutiver Organisationsform oder solche, die mit VSAM organisiert sind. Es ist möglich, für ii einen Wert zu benennen, der größer ist als es eigentlich erforderlich wäre.

Zur bloßen Übersetzung eines Programmes ist die folgende Anordnung der Steuerkarten zu treffen:

```
// JOB jobname
  (OPTION-Karten im Bedarfsfall)
// EXEC übersetzername
  (Quellenprogramm)
/*
/&
```

Die Zeichenfolge /* auf den Spalten 1 und 2 zeigt das Datenende an. Wenn das Lesen über SYSIPT oder SYSRDR erfolgt, bewirkt /* , daß der Supervisor die Dateiendebedingung an das Problemprogramm weiterleitet, außerdem wird sie auch festgehalten. Streng gesprochen kann / weggelassen werden, wenn darauf eine Steuerkarte mit /& folgt. Es zeugt jedoch von einer praktischen Gewissenhaftigkeit, wenn stets beide Steuerkarten eingefügt werden.

1) Auch als „Linkage Editor" bezeichnet

2) LBLTYP ist aus der Zusammenziehung von LABEL TYP (Kennsatztyp) entstanden.

Ein einfacher Job kann viele Dinge einschließen. Betrachten wir einmal das nachfolgende Beispiel:

```
// JOB VIELAUF
// EXEC ASSEMBLY
  (Quellenprogramm in der Assemblierersprache)
/*
// OPTION LISTX
// EXEC FCOBOL
  (Quellenprogramm in COBOL)
/*
// EXEC FCOBOL
  (Quellenprogramm in COBOL)
/*
/&
```

Der soeben vorgestellte Job ist aus drei Jobsteps zusammengesetzt. Ein einmal gelesenes OPTION-Statement wird auf alle bis zum Jobende nachfolgenden Jobsteps angewendet, wenn nicht eine gegensätzliche Option auf einer anderen, später erscheinenden OPTION-Steuerkarte vermerkt ist.

Um ein Programm zu übersetzen, zu binden und danach sofort auszuführen, müssen wir ein Kartenpaket zusammenstellen, das folgendes Aussehen aufweist:

```
// JOB jobname
// OPTION LINK
  (Steuerkarte PHASE, falls notwendig)
// EXEC übersetzername
  (Quellenprogramm)
/*
  (Steuerkarte LBLTYP, falls erforderlich)
// EXEC LNKEDT
  (Steuerkarten für Kennsätze, ASSGN- und OPTION-Steuerkarten,
                                        falls erforderlich)
// EXEC ,SIZE=AUTO
  (Datenkarten für das Benutzerprogramm mit abschließender
            Steuerkarte /*, falls Daten über diese Eingabeeinheit eingelesen werden müssen)
/&
```

Der SIZE-Operand kann entfallen, wenn kein virtueller Speicherplatz über GETVIS angefordert wird, d.h. wenn weder VSAM benutzt wird noch in einem Programm in der Assemblierersprache mit dem Makro GETVIS gearbeitet wird.

Die Steuerkarte

```
// OPTION LINK
```

wird immer dann benötigt, wenn die Standardoption nicht LINK ist; diese Option aber wird kaum bei der Generierung eines Systems berücksichtigt. Die Option LINK bedeutet, daß die Ausgabe des übersetzten Programmes auf der logischen Einheit SYSLNK aufgezeichnet wird, von der der Binder anschließend dieses Programm liest.

Unter einer Phase versteht man alle Programmteile (Programmeinheiten) eines Programmes, die in einem bestimmten Augenblick der Ausführung zusammenwirken. Durch Austausch von Programmeinheiten (Unterprogrammen etc.) während des Ablaufes entstehen die verschiedenen Phasen eines Programmes. Durch dieses Überlagerungsprinzip können Einsparungen an Hauptspeicherplatz erzielt werden, da ein oder mehrere neue Programmeinheiten denselben Platz wie die ausgetauschte Programmeinheit bzw.

die ausgetauschten Programmeinheiten einnehmen können. Ein Programm kann eine oder mehrere Phasen aufweisen.

Die Steuerkarte PHASE besitzt folgenden Aufbau:

```
PHASE phasenname,operand
```

Der Phasenname ist wie der Jobname aus 1 bis 8 Zeichen zu bilden, wobei das erste ein Buchstabe sein muß, die restlichen können Buchstaben oder Ziffern sein. Gewöhnlich wird als Operand einer der folgenden Ausdrücke verwendet:

```
*
*±verschiebung
symbol
symbol±verschiebung
S
ROOT
```

Bei Angabe von S bezieht man sich auf den Beginn der virtuellen Speicherpartition. Das Sternzeichen verweist auf den ersten freien Speicherplatz hinter irgendeiner Phase des gleichen Programmes. Wenn in einem Job der Operand * auf der ersten Phasenkarte erscheint, so sind *und S äquivalente Operanden. Unter verschiebung muß entweder eine ein- bis achtstellige Dezimalzahl oder eine Sedezimalzahl in der Form X'ssssss', wobei unter s Sedezimalziffern verstanden werden sollen, erscheinen. Die Sedezimalzahl muß nicht unbedingt sechs Sedezimalziffern aufweisen, ihre Stellenzahl kann zwischen 1 und 6 liegen. Wird symbol als Operand benutzt, so muß es sich um einen externen Namen handeln, der in einer früheren, im gleichen Job verarbeiteten Phase definiert ist. ROOT wirkt wie S mit der Ausnahme, daß es die Einschränkung auferlegt, daß keine nachfolgende Phase irgendeinen Teil dieser Wurzelphase [1] überlagern darf.

Wenn wir die Steuerkarte

```
// OPTION CATAL
```

an Stelle der Steuerkarte

```
// OPTION LINK
```

verwenden, dann wird nach dem Lesen der Steuerkarte /& für das Jobende das gebundene Programm in der Bibliothek der ausführbaren Programme (Bibliothek der Lademoduln) unter dem angegebenen Phasennamen katalogisiert. Ein Programm, das unter dem gleichen Phasennamen bereits in der Bibliothek vorhanden ist, wird dabei rigoros zerstört. Wenn ein Programm aus mehr als einer Phase besteht, so müssen die ersten vier Zeichen der Namen aller Phasen dieses Programmes die gleichen sein.

Um ein in die Bibliothek der Lademoduln aufgenommenes Programm auszuführen, ist das folgende Steuerkartenpaket zusammenzustellen:

```
// JOB jobname
  (Steuerkarten für Kennsätze, ASSGN- und OPTION-Steuerkarten,
                                        falls erforderlich)
// EXEC phasenname
  (Datenkarten für das Benutzerprogramm mit abschließender
          Steuerkarte /*, falls Daten über diese Eingabeein-
          heit eingelesen werden müssen)
/&
```

1) Im angloamerikanischen Sprachraum als Rootphase bezeichnet

Eine Prozedurbibliothek kann auf der SEDZ eingerichtet werden, die als Systemresidenz dient. In ihr werden sogenannte Prozeduren untergebracht. Unter einer Prozedur versteht man in diesem Zusammenhang eine Menge von in der Jobsteuersprache [1] formulierten Steuersätzen (gegebenenfalls auch Datensätzen), die vom Systemprogrammierer [2] im voraus definiert wurden. Zur Bezugnahme auf eine solche vorgefertigte Prozedur genügt die Niederschrift des Steuerstatements

```
// EXEC PROC=prozedurname
```

Der Prozedurname wird beim Einspeichern der Prozedur in die Bibliothek vergeben. Beim Auftauchen eines solchen Steuerstatements veranlaßt das Steuerprogramm die Aufnahme aller unter prozedurname verfügbaren Statements an Stelle des o.a. EXEC-Statements in den betreffenden Job. In den Prozeduren selbst können wiederum ein oder mehrere EXEC-Statements stehen, die sich dann natürlich auf bestimmte Programme konzentrieren müssen. Auch die Operanden REAL und SIZE können dabei angegeben werden. Der Hauptzweck beim Einsatz von Prozeduren ist wohl darin zu sehen, daß sie die Vorbereitung von Kartenpaketen für die Zusammenstellung, den Aufbau und den Ablauf von Jobs wesentlich vereinfachen.

10.9. Die Steuerstatements für die Standardkennsätze im Betriebssystem DOS/VS

Für jede einem Magnetband zugeordnete Datei, die mit Standardetiketten (Standardkennsätzen) von einem Programm verarbeitet werden soll, muß ein TLBL-Steuerstatement [3] dem EXEC-Statement vorausgehen. Ein solches Statement lautet allgemein:

```
// TLBL  dateiname,['datenmengenname'],[datum],
         [reihennummer],[datenträgernummer],
         [dateinummer],[generation],[versionsnummer]
```

Mit Ausnahme von dateiname ist die Verwendung aller anderen Angaben im TLBL-Statement freigestellt. Wenn der Datenmengenname aufgeführt wird, so ist er in Anführungszeichen einzuschließen; fehlt er, so wird der Dateiname gleichzeitig auch als Datenmengenname genommen. Bekanntlich bezieht man sich in einem Quellenprogramm auf die anzusprechende Datenmenge mittels des Dateinamens.

Unter datum ist bei Ausgabedateien das Verfalldatum, bei Eingabedateien das Erstellungsdatum anzugeben. Das Datum ist dabei in der Form jj/ttt (jj Jahr, ttt laufender Tag des Jahres) zu spezifizieren. Bei der Ausgabe kann an Stelle dieser Form auch die ein- bis vierziffrige Schreibweise tttt benutzt werden. Damit wird ausgesagt, daß das Verfalldatum tttt Tage nach dem augenblicklichen Datum liegt. Wenn bei der Eingabe das Erstellungsdatum nicht angegeben ist, so unterbleibt seine Nachprüfung. Wird bei Ausgabedateien die Datumsangabe weggelassen, so wird eine Sperrfrist von 0 Tagen vorgesehen.

Die letzten fünf Operanden beziehen sich auf bestimmte Felder des Bandkennsatzes. Fehlen sie bei Eingabedateien, so wird die Prüfung des entsprechenden Feldes unterdrückt. Sind diese Operanden bei Ausgabedateien weggelassen, so werden sie durch die in Abb. 10.1. aufgeführten Standardangaben ersetzt.

1) Alle Steuerstatements werden als Statements einer Sprache betrachtet, die als Jobsteuersprache (englisch: Job Control Language, abgekürzt JCL) bezeichnet wird

2) Systemprogrammierer sind Mitarbeiter, zu deren Pflichtenkreis die Pflege und Wartung des Betriebssystemes der betreffenden Installation gehört

3) TLBL ist aus der Zusammenziehung von Tape Label (Bandkennsatz) entstanden

Angabe (Operand)	Bedeutung	Standard
reihennummer	Kennung des Datenträgers	Datenträgerkennung des ersten Datenträgers, d.h. hier der ersten Magnetbandrolle
datenträgernummer	Laufende Nummer des Datenträgers	ØØØ1
generation	Generationsnummer	ØØØ1
versionsnummer	Versionsnummer	Ø1

Abb. 10.1. Standardangaben an Stelle der Angaben im TLBL-Steuerstatement

Wenn in einem Programm auf Dateien zugegriffen wird, deren Daten auf SEDZ gespeichert sind, so müssen dem EXEC-Steuerstatement DLBL- und EXTENT-Steuerkarten [1] vorausgehen. Diese beiden Kartenarten sind wie folgt aufgebaut:

```
// DLBL  dateiname, ['datenmengenname'] ,[datum],[schlüssel]
// EXTENT  [SYSxxx],[reihennummer],[typ ],[folge],
           [relspur],[spurzahl],[spurzyl]
```

Die ersten drei Operanden des DLBL-Statements haben die gleiche Bedeutung wie die entsprechenden Operanden des TLBL-Statements. Eine Ausnahme ist jedoch zu beachten: Fehlt bei Ausgabedateien der Operand datum, so wird eine Sperrfrist von 7 Tagen unterstellt. Unter schlüssel ist eine der in Abb. 10.2. dargestellten Angaben aufzuführen.

Schlüssel	Bedeutung
SD	Konsekutive Organisationsform
DA	Regionale Organisationsform (DA-Direct Access)
ISC	Indexsequentielle Organisationsform: Erstellung der Datenmenge (C-Creation)
ISE	Indexsequentielle Organisationsform: Existierende Datenmenge (E-Existing)
VSAM	Organisationsform gemäß VSAM

Abb. 10.2. Alternativen für den Schlüsseloperanden im DLBL-Statement

Wenn der Schlüssel für die Zugriffsmethode nicht angegeben ist, so wird SD angenommen.

Alle Operanden der EXTENT-Statements sind Optionen, d.h. sie können, aber müssen nicht aufgeführt werden. Werden alle Operanden weggelassen, so kann das EXTENT-Statement selbst dann entfallen, wenn die Datenmenge nicht mehr als einen Bereich auf einer SEDZ belegt. Belegt eine Datenmenge mehrere Bereiche, so ist für jeden Bereich ein gesondertes EXTENT-Statement zu formulieren. Die Reihenfolge der Statements entspricht dabei der fortlaufenden Bereichsnummer (siehe Angabe folge).

1) DLBL ist aus der Zusammenziehung von Disk Label (Plattenkennsatz) entstanden

Durch SYSxxx wird die logische Einheit festgelegt, auf der der Bereich liegt; gibt es nur einen Bereich und ist zusätzlich im Programm SYSxxx festgelegt, so kann dieser Operand entfallen. Wenn sowohl im Programm als auch im EXTENT-Statement die Angabe von SYSxxx erfolgt, so wird die Angabe im Programm ignoriert. Für VSAM muß SYSxxx auf jeden Fall aufgeführt werden.

Unter reihennummer ist die Kennung des Datenträgers aufzuführen, zu dem der Bereich gehört. Wird reihennummer weggelassen, so wird für reihennummer die Angabe aus dem vorhergehenden EXTENT-Statement genommen. Ist allerdings nirgendwo reihennummer aufgeführt, also in keinem der vorhergehenden Statements, so unterbleibt die Prüfung der Kennung des Datenträgers. Für Datenmengen auf VSAM-Basis muß dieser Operand vorhanden sein.

Unter typ kann eine der folgenden Angaben vorgenommen werden:

1	Datenbereich, jedoch nicht für eine Datenmenge, die einen Zylinder mit einer oder mehreren anderen Datenmengen teilt
2	Überlaufbereich für eine logisch fortlaufend organisierte Datenmenge
4	Bereich für das Inhaltsverzeichnis (den Index) bei einer logisch fortlaufend organisierten Datenmenge
8	Datenbereich für eine Datenmenge, die sich einen Zylinder mit einer oder mehreren anderen Datenmengen teilt (konsekutive Organisationsform)

Unter einer Datenmenge, die sich einen Zylinder mit einer oder mehreren anderen Datenmengen teilt, versteht man eine Datenmenge, für die nur bestimmte Schreib/Lese-Köpfe agieren, z.B. die Schreib/Lese-Köpfe 0,1,2 und 3 der Zylinder 120 bis 125. Wird der Operand typ weggelassen, so wird standardmäßig 1 angenommen.

Mit folge ist die fortlaufende Bereichsnummer gemeint. Für den ersten Bereich einer Datenmenge ist 0 anzugeben, für den zweiten 1 usw. Wird dieser Operand weggelassen, so wird für folge 0 angenommen. Bei indexsequentiell organisierten Datenbeständen ist eine Ausnahme zu berücksichtigen: Der Hauptindex bekommt die Bereichsnummer 0, der Zylinderindex 1, die Bereiche zur Aufnahme der Daten werden ab 2 gezählt. Existiert kein Hauptindex, so gibt es auch keinen Bereich mit der Nummer 0. Dieser Operand wird bei Datenmengen, die der indexsequentiellen Organisationsform unterliegen, unbedingt benötigt.

Unter relspur ist ein- bis fünfstellig relativ zur Spur 0 des Zylinders 0 die Spur festzulegen, auf der die Datenmenge beginnt. Zum Beispiel ist für eine Datenmenge, die auf Zylinder 80 die Spuren 3 bis 7 belegt, als relative Spurnummer 1523, errechnet aus 80·19+3, anzugeben, wenn das Magnetplattenspeichergerät IBM 3330 als Träger für diese Datenmenge dient. Bei konsekutiv, regional und auf VSAM-Basis organisierten Datenmengen kann dieser Operand ausgelassen werden; die erforderlichen Informationen werden dann aus dem Kennsatz des Formates 1 extrahiert.

Unter spurzahl ist ein- bis fünfstellig die Anzahl der Spuren aufzuführen, die zum betreffenden Bereich der Datenmenge gehören. Diese Angabe kann für vorhandene Datenmengen entfallen, wenn sie konsekutiv oder regional oder auf VSAM-Basis organisiert sind.

Ein- oder zweistellig ist unter spurzyl die obere Spurnummer für eine konsekutiv organisierte Datenmenge festzulegen, die sich einen Zylinder mit einer oder mehreren anderen Datenmengen teilt. Wenn z. B. eine Datenmenge die Spuren 3 bis 7 auf den

Zylindern 80 bis 88 einer IBM 3330 belegen soll, d.h. daß für sie nur ein Bereich reserviert werden soll, so ist das EXTENT-Steuerstatement wie folgt zu formulieren:

```
// EXTENT  SYSØØ6,111111,8,1523,45,7
```

Kennsatzinformationen können bei Datenmengen weggelassen werden, für die solche Informationen ständig auf der Systemresidenz gespeichert sind. Eine solche Sachlage liegt in der Regel bei den für die Übersetzer eingerichteten Arbeitsdateien vor.

10.10. Das Systemprogramm POWER im Betriebssystem DOS/VS

Das Betriebssystem DOS/VS unterstützt den Mehrprogrammbetrieb [1] von bis zu fünf nicht aufeinander bezogenen Programmen. Jedes Programm läuft dabei in einer sogenannten Partition ab, die wenigstens 64 k virtuellen Speicher und mit Ausnahme des für den Seitenwechsel erforderlichen sonst keinen weiteren Realspeicher umfassen braucht. Der Umfang der Partitionen wird durch den Operateur festgesetzt.

Jedes Problemprogramm erfordert die Zuweisungen von SYSRDR und SYSLST. Wenn nun fünf Programme zur gleichen Zeit ausgeführt werden sollen, so resultiert daraus die Erfordernis, daß fünf Kartenleser und Drucker vorhanden sind. Bei einer Installation mit dem Betriebssystem DOS eine Maschinenausrüstung in einer solchen Fülle zu erwarten, erscheint uns reichlich utopisch, ist doch das DOS zur Befriedigung der Bedürfnisse kleinerer Installationen konzipiert worden. Welche Möglichkeiten zeichnen sich nun zur Lösung des soeben umrissenen Problems ab. Da bestände zunächst einmal die Möglichkeit, in einer Partition ein Dienstprogramm einzusetzen, das Kartenpakete von einem Lochkartenleser auf ein Magnetband oder einen Plattenstapel überträgt. Bei den in den anderen Partitionen laufenden Programmen müßten dann SYSRDR und SYSLST einem Magnetband- bzw. einem Magnetplattenspeichergerät zugewiesen werden. Eine solche Jobabfertigung wäre äußerst unhandlich, vor allem dort, wo sehr viele Jobs während einer Schicht durchzuziehen sind. Eine andere Lösung bietet sich durch den Einsatz des Programmes POWER an.

Wird POWER eingesetzt, so läuft es als eines der für den Mehrprogrammbetrieb zulässigen bis zu fünf Programme. Sein Speicherbedarf liegt bei 20 k Realspeicher im Minimum. Bei normaler Arbeitsweise liest es Statements und Daten über die logische Systemeinheit SYSRDR ein und kopiert sie in eine spezielle Datei auf einem Magnetplattenspeicher. Die so entstandene Datenmenge wird als Spool-Datenmenge [2] bezeichnet. Die von der Arbeitsvorbereitung zusammengestellten Jobs werden solange zunächst einfach hintereinander eingelesen, bis der für die Spool-Datenmenge vorgesehene Platz auf dem Plattenspeicher erschöpft ist. Nach dem Einlesen steht der Job ausführbereit und der Operateur kann seine Ausführung in irgendeiner verfügbaren Partition veranlassen.

Das Problemprogramm weist SYSRDR nicht einem wirklichen Kartenleser zu, sondern zu einem Scheingerät, d.h. zu einer Kanalgeräteadresse, die kein tatsächlich an das DV-System angeschlossenes Gerät repräsentiert. Versucht nun das Problemprogramm von SYSRDR einzulesen, so arbeiten Supervisor und POWER zusammen, um die Anforderung abzufangen und den Satz aus dem Spoolbereich zur Verfügung zu stellen. In gleicher Weise werden den Einheiten SYSPCH und SYSLST Scheinadressen zugewiesen

---

1) Auch als „Multiprogramming" bezeichnet

2) Spool ist die Abkürzung von „simultaneous peripheral operations on line". Man versteht darunter eine Datenmenge, die von einem Programmsteuersystem erzeugt wurde, das unabhängige EA-Operationen während der Zeit ausführt, in der ein oder mehrere andere Programme parallel dazu ablaufen.

und die für sie vorgesehene Ausgabe wird durch POWER auf einen Magnetplattenstapel oder ein Magnetband geschrieben. Nach der Beendigung eines Jobs veranlaßt POWER die endgültige Ausgabe aufs Zielgerät sobald die für POWER bereitgestellten Einheiten SYSPCH bzw. SYSLST frei und unbeschäftigt sind.

Der Programmierer braucht nicht zu wissen, daß POWER eingesetzt wird, da der Programmierer in der Regel nicht die ASSGN-Befehle oder -Statements für die logischen Systemeinheiten benutzt oder abfaßt.

10.11. Der Informationsaustausch zwischen Jobsteps beim Betriebssystem DOS/VS

Für jede Partition, die für Problemprogramme eingerichtet wird, sieht das Betriebssystem DOS ein zwölf Bytes großes Gebiet vor, das zum Informationsaustausch zwischen Jobsteps (Austauschgebiet für den Benutzer) benutzt werden kann. Zu Beginn eines Jobs wird jede Bitstelle dieses Gebietes auf Null gelöscht. Das 12. Byte dieses Gebietes ist der sogenannte Prüfanzeiger für das Benutzerprogramm [1]. Ein spezielles Statement der Jobsteuersprache kann dazu benutzt werden, um diesem Byte einen Wert zuzuweisen. Dieses Statement lautet:

```
// UPSI xxxxxxxx
```

Jedem Zeichen x im Operanden entspricht gerade ein Bit des UPSI-Bytes. Wird für x die Ziffer 0 angegeben, so wird die betreffende Bitstelle auf 0 gesetzt. Bei x gleich 1 erfolgt die Zuweisung einer 1 zur Bitstelle. Wird für x der Buchstabe X angegeben, so bleibt die Bitstelle ungeändert. Werden weniger als acht Zeichen im UPSI-Statement aufgeführt, so werden nach rechts X ergänzt.

Stellen (Bytes)	Inhalt
0 bis 7	Datum in der Form jj/mm/tt (Jahr, Monat, Tag) oder in der Form tt/mm/jj. Die benutzte Form hängt von der zur Systemgenerierungszeit getroffenen Vorkehrung in der betreffenden Installation ab.
8 bis 11	Nicht vorausschaubar
12 bis 22	Erste elf Bytes des Informationsaustauschgebietes für den Benutzer
23	UPSI
24 bis 31	Jobname aus dem JOB-Steuerstatement
32 bis 35	Adresse des letzten Bytes der Partition
36 bis 39	Adresse des letzten Bytes der augenblicklich ablaufenden Programmphase
40 bis 43	Adresse des letzten Bytes, das von irgendeiner Programmphase belegt wird
44 und 45	Länge des Merkmalgebietes des Programmes

Abb. 10.3. Gebiet für den Informationsaustausch

1) Im angloamerikanischen Sprachraum als „User Program Sense Indicator", abgekürzt UPSI bezeichnet.

Das zwölf Bytes umfassende Austauschgebiet für Jobsteps ist Teil eines größeren Gebietes, auf dessen Inhalt der Programmierer zugreifen kann. In der Abb. 10.3. ist der Inhalt der ersten 46 Bytes dieses Gebietes aufgeführt.

Aufgabengruppe 23

1. Klassifiziere jede der nachstehenden Zeichenfolgen in Hinblick auf das Betriebssystem DOS als
   a) eine logische Systemeinheit
   b) eine logische Programmiereinheit
   c) sonstige

   Zu klassifizieren sind:
   ① SYSRDR
   ② Ø14
   ③ SYSØØ7
   ④ SYS346

2. Klassifiziere die nachfolgenden Steuerstatements bzw. Operateurbefehle im Hinblick auf das Betriebssystem DOS als
   a) permanente Gerätezuweisung
   b) temporäre Gerätezuweisung
   c) ungültig

   Zu klassifizieren sind

```
①      // ASSGN  SYSØ14,X'Ø14'
②      // ASSGN  SYSRDR,IGN
③      // ASSGN  SYS288,X'1AØ'
④      // ASSGN  SYSUSE,X'281'
⑤         ASSGN  SYSIN,X'28Ø'
```

3. Jedes der nachfolgenden Steuerstatements der Jobsteuersprache für das Betriebssystem DOS ist ungültig. Die Statements sind in geeigneter Weise zu berichtigen!

```
a)    //EXEC  PLIOPT
b)    //  EXEC  LINKEDITOR
c)    //  UPSI  1ØØØ1ØØ1ØØ
d)        UPSI  1Ø1X1
e)    //  OPTIONS  LINK,DUMP
```

4. In welcher Bibliothek des DOS sind folgende Informationspakete zu erwarten?
   a) PL/1-Kompilierer
   b) Supervisor
   c) Ausgabe des Binders (Linkage Editors)
   d) Modul, das als Eingabe für den Binder dient

5. Die Eintragungen in den beiden Spalten der nachstehenden Tabelle sind so zu ordnen, daß die Eintragung in Spalte 2 der Eintragung in Spalte 1 angepaßt ist.

Fall	Art des Statements (Spalte 1)	Bedeutung (Spalte 2)
a	JOB-Steuerstatement	Markierung des Jobendes
b	EXEC-Steuerstatement	Zurücksetzung des UPSI-Bytes auf Null
c	/*-Statement	Markierung des Endes einer Datenmenge
d	/&-Statement	Markierung des Beginns eines Jobsteps

10.12. Konzepte des Betriebssystems OS

In diesem und in den folgenden vier Abschnitten wollen wir so viele Informationen kennenlernen, wie notwendig sind, um Steuerstatements für die Programme erstellen zu können, die man höchstwahrscheinlich immer dann einsetzen wird, wenn man programmieren lernen will.

Das Betriebssystem OS erfordert das Vorhandensein mehrerer Datenmengen bzw. Bibliotheken. Wir werden besonders über die in Abb. 10.4. aufgeführten zu diskutieren haben.

	Datenmengenname	Inhalt
Allgemeine Datenmengen	SYS1.SVCLIB	SVC-Bibliothek
	SYS1.LINKLIB	Bibliothek für den Binder
	SYS1.PROCLIB	Prozedurbibliothek
	SYS1.IMAGELIB	Abbildungsbibliothek
	SYS1.STGINDEX	Bibliothek für virtuelle EA
	SYS1.PLILIB	Bibliothek für PL/1
	SYS1.FORTLIB	Bibliothek für FORTRAN
	SYS1.COBLIB	Bibliothek für COBOL
	SYS1.MACLIB	Makrobibliothek
Spezielle Datenmengen für VS1	SYS1.PAGE	Seitenbibliothek
	SYS1.SYSJOBQUE	Jobwarteschlange
	SYS1.SYSPOOL	Spooldatenmenge
	(beliebig)	Hauptkatalog für VSAM
	SYSCTLG	Systemkatalog

Abb. 10.4. Spezielle Datenmengen bzw. Bibliotheken im Betriebssystem OS

Die letzten fünf in der Aufstellung von Abb. 10.4. aufgeführten Datenmengen können für das Betriebssystem OS/VS1 zusätzlich erforderlich werden.

Bei der Variante VS1 des Betriebssystemes OS enthält SYS1.PAGE den virtuellen Speicher des Systems. Bei der Variante VS2 werden hierzu mehrere Datenmengen benutzt, deren Namen von den Systemprogrammierern festgelegt werden können. Systemprogrammierer sind Fachkräfte, die verantwortlich für die Pflege und Wartung des Betriebssystemes sind.

Im OS/VS1 wird die Warteschlange der zur Ausführung bereitstehenden Jobs in der Datenmenge SYS1.SYSJOBQE festgehalten; im OS/VS2 wird diese Warteschlange in einem Gebiet des virtuellen Speichers untergebracht. Die von den beiden Varianten des Betriebssystemes OS benötigten Arbeitsdateien werden selbstverständlich auf SEDZ [1] angelegt.

Bei der Variante VS1 werden Datenmengen, die im Eingabestrom zusammen mit Steuerstatements erscheinen, sowie Datenmengen, die gedruckt oder gestanzt werden sollen, in SYS1.SYSPOOL festgehalten. Bei der Variante VS2 dienen ein oder mehrere Datenmengen mit frei gewählten Namen dem gleichen Zweck. Die meisten der generierten VS1-Systeme sowie alle VS2-Systeme enthält der Hauptkatalog für VSAM Informationen über das EA-Gerät, den Typ und die Datenträgerseriennummer von ausgewählten Datenmengen. Bei manchen Datenmengen kann er darüberhinaus zusätzliche Informationen aufweisen, die normalerweise nicht im Kennsatz des Formates 1 des Datenträgerinhaltsverzeichnisses (VTOC) gefunden werden. SYSCTLG enthält die gleichen Informationen bei VS1-Systemen, die VSAM nicht unterstützen. Wir wollen zukünftig den Begriff Katalog immer dann verwenden, wenn wir entweder den Hauptkatalog von VSAM oder SYSCTLG meinen.

Weitere Bibliotheken können jederzeit in Installationen eingerichtet werden. Viele Benutzer richten insbesonders Bibliotheken für ihre eigenen Zwecke ein; wir wollen sie als Privatbibliotheken bezeichnen. Datenmengen, die für mehr als einen Job Gültigkeit haben, die also die Jobgrenzen sprengen, gelten als permanente Datenmengen. Datenmengen, die im gleichen Job sowohl erzeugt als auch wieder zerstört werden, heißen temporäre Datenmengen. Wie hier eingeführt, schließt der Begriff Job oder Auftrag irgendeine Menge von Programmen ein, die zwecks ihrer Ausführung durch den Computer zusammengefaßt werden.

In die Ausführung eines Jobs sind folgende Komponenten verwickelt:

1. Ein Spezialprogramm, Systemleser [2] genannt, liest die Jobsteuerstatements von einem durch die Maschinenbedienung festgelegten Standardeingabegerät, das als SYSIN bezeichnet wird. Die Jobsteuerstatements teilen dem Betriebssystem alle in den Job eingehenden Programme und Datenmengen mit. Nach der Aufbereitung werden die Jobsteuerstatements in die Jobwarteschlange gestellt. Mitunter sind auch Datenmengen in dem über SYSIN einlaufenden Eingabestrom vorhanden; diese werden in die Spooldatenmenge hineingestellt.
2. Danach kommt ein anderes Systemprogramm, ein Initiator, zum Zuge. Es untersucht zunächst, ob die benötigten EA-Geräte verfügbar sind. Wenn dem so ist, so ordnet es die Hilfsmittel des Systems, die Systemelemente, zu, damit die Anforderungen des Programms bezüglich der Datenmengen erfüllt werden können. Das erste Programm des Jobs wird in den Hauptspeicher geladen und ihm die Programmablaufsteuerung übertragen.

---

1) SEDZ ist die Abkürzung von Speichereinheiten mit direkter Zugriffsmöglichkeit

2) Im angloamerikanischen Sprachraum als System Reader bezeichnet

3. Das Programm des Benutzers wird ausgeführt. Wir wollen es Problemprogramm nennen, um es ganz klar von den speziellen Systemprogrammen zu unterscheiden. Nach seiner Beendigung übergibt es die Programmablaufsteuerung zurück an den Supervisor, der diese wiederum dem Initiator überträgt. Die zu druckenden Ausgabedaten werden gewöhnlich in die Spooldatei geschrieben, die als SYSOUT-Datenmenge bezeichnet wird. Problemprogramme drucken die Daten direkt nur in den seltesten Fällen, d.h. nach entsprechender Vorkehrung in den Jobsteuerstatements.

4. Wenn dem ersten ausgeführten Programm innerhalb des Jobs weitere folgen müssen, d.h. der Job setzt sich aus mehreren Jobsteps zusammen, so werden die Pkt. 2. und 3. für jeden Jobstep wiederholt. Nach Beendigung des letzten Programmes bringt der Initiator alle SYSOUT-Datenmengen in die Warteschlange des Ausgabeschreibers [1], die Teil der Jobwarteschlange ist.

5. Ein spezielles Systemprogramm, Ausgabeschreiber [1] genannt, liest die SYSOUT-Datenmengen und erledigt das tatsächliche Drucken. Wenn es gewünscht wird, kann die Ausgabe auch auf andere Datenträger erfolgen (Stanzen von Lochkarten, Aufzeichnen auf Magnetbänder).

Die Organisation des Realspeichers und des virtuellen Speichers differiert etwas zwischen den beiden Betriebssystemvarianten VS1 und VS2. Bei beiden enthält der untere Teil des Realspeichers den Supervisor und verschiedene Arbeitsgebiete. Der obere Teil des virtuellen Speichers nimmt Routinen des Supervisors auf, die nicht die ganze Zeit im Realspeicher liegen müssen sowie einige Arbeitsgebiete, die durchaus ebenfalls dem Seitenwechsel unterzogen werden können. Der dazwischen befindliche Speicherteil steht für Problemprogramme zur Verfügung. Bei der Variante VS1 wird der verfügbare virtuelle Platz in sogenannte Partitionen eingeteilt. Jeder Partition wird Platz im Vielfachen von 64 k zugeordnet. Maximal sind 15 Partitions erlaubt, doch die meisten Systeme weisen nicht mehr als fünf auf. Ein Partitionumfang von 128 k dürfte ausreichen, um jedes Programm aufzunehmen, von dem in diesem Buch gesprochen wird. Wenn die Maschinenbedienung die Partitionen definiert, so weist sie ihnen nicht nur ihre Größe zu, sondern sie legt auch die Jobklassen fest, die die Partitionen benutzen können. Die Jobklassen reichen von A bis inklusive O. Eine Partition kann eine oder mehrere Jobklassen bedienen. Bei VS2 werden jeder Partition $2^{24} \approx 16$ Millionen Bytes zugewiesen, abzüglich freilich der Systemerfordernisse im unteren und oberen Speicherbereich. Unter diesem Aspekt ist es wohl klar, daß eine größere Seitendatei als bei VS1 vorhanden sein muß, doch VS2 ist nun einmal für sehr große Systeme entworfen worden. Der Schutz gegenüber unberechtigten Einwirkungen eines Programmes auf Daten und Instruktionen anderer Programme wird beim OS/VS2 durch passende Handhabung der Segment- und Seitentabellen erreicht.

Während oder nach dem einleitenden Programmladen (IPL) gibt der Operateur eine Reihe von Startbefehlen ein. Wir wollen sie kurz besprechen:

a) START RDR
Dieser Startbefehl bewirkt, daß der Systemleser mit dem Lesen des Eingabestroms (Steuerstatements und dazwischen geschobene Daten) beginnt.

b) START INIT
Dieser Startbefehl veranlaßt, daß die Initiatoren mit ihrer Arbeit beginnen. Beim OS/VS1 wird dieser Befehl auf die Partition bezogen, die für Programme zugänglich gemacht werden soll. Beim OS/VS2 bezieht sich dieser Befehl auf einen Namen, un-

1) Im angloamerikansichen Sprachraum als Output Writer bezeichnet

ter dem das System den Initiator und die Jobklassen, die der betreffende Initiator verarbeiten soll, identifizieren kann.

c) START WTR
Dieser Befehl veranlaßt den Arbeitsbeginn der Systemausgabeschreiber. Er bezieht sich auf die Adresse des Druckers, Stanzers oder Magnetbandgerätes und die Ausgabeklasse [1], die der Schreiber verarbeiten soll. Es existieren 36 Ausgabeklassen, A bis Z sowie 0 bis 9. Die Standardübereinkunft sieht vor, daß die Ausgabeklasse A für die Druckausgabe und die Klasse B für die Lochkartenausgabe reserviert ist.

Für die Systemleser, die Systemschreiber und im allgemeinen auch für die meisten anderen EA-Anforderungen kann ein EA-Gerät auf drei verschiedene Weisen zugewiesen werden:

1. Durch Angabe der *Kanalgeräteadresse*, z.B. durch ØØE, 15Ø usw.
2. Durch einen *Gattungsnamen*, z.B. 14Ø3, 333Ø usw. Der Gattungsname ist auch als Gerätetypbezeichnung bekannt. Die nachfolgende Zusammenstellung zeigt, wie die möglichen Angaben interpretiert werden.

24ØØ	Irgendein Magnetbandlaufwerk der Serien IBM 2400 oder IBM 3400 mit einer Aufzeichnungsdichte von 800 bpi [2] ohne verschiedene Aufzeichnungsdichten bzw. von 800 und 1600 bpi mit verschiedener Aufzeichnungsdichte.
24ØØ-3	Irgendein Magnetbandlaufwerk der Serien IBM 2400 oder IBM 3400, die nur die Aufzeichnungsdichte 1600 bpi besitzen.
24ØØ-4	Irgendein Magnetbandlaufwerk der Serien IBM 2400 oder IBM 3400 mit verschiedener Aufzeichnungsdichte, d.h. sowohl mit der Aufzeichnungsdichte 800 bpi als auch mit der Aufzeichnungsdichte 1600 bpi.
34ØØ-3	Irgendein Magnetbandlaufwerk der Serien IBM 3410 oder IBM 3420 mit der alleinigen Aufzeichnungsdichte von 1600 bpi
34ØØ-4	Irgendein Magnetbandlaufwerk der Serien IBM 3410 oder IBM 3420 mit den beiden Aufzeichnungsdichten 800 und 1600 bpi.
34ØØ-5	Irgendein Magnetbandlaufwerk des Typs IBM 3420 mit der alleinigen Aufzeichnungsdichte von 6250 bpi.
34ØØ-6	Irgendein Magnetbandlaufwerk des Typs IBM 3420 mit den beiden verschiedenen Aufzeichnungsdichten 6250 bpi und 1600 bpi.
23Ø5-1	Irgendein Magnetplattenspeichergerät IBM 2305, Modell 1
23Ø5-2	Irgendein Magnetplattenspeichergerät IBM 2305, Modell 2
2314	Irgendein Laufwerk eines Magnetplattenspeichergerätes der Typen IBM 2314 oder IBM 2319.
333Ø	Irgendein Laufwerk eines Magnetplattenspeichergerätes der Typen IBM 3330, Modell 1, oder IBM 3333, Modell 1.
333Ø-1	Irgendein Laufwerk eines Magnetplattenspeichergerätes der Typen IBM 3330, Modell 11,oder IBM 3333, Modell 11.
334Ø	Irgendein Laufwerk eines Magnetplattenspeichergerätes des Typs IBM 3340.

1) Im angloamerikanischen Sprachraum als Output Class bezeichnet
2) Abkürzungen von Bytes/inch, d.h. Anzahl der auf ein Zoll Bandlänge aufgezeichneten Bytes

14Ø3	Irgendein Zeilendrucker des Typs IBM 1403.
1443	Irgendein Zeilendrucker des Typs IBM 1443, Modell N1.
3211	Irgendein Zeilendrucker des Typs IBM 3211.
25Ø1	Irgendein Kartenleser des Typs IBM 2501.
252Ø	Irgendein Lochkartengerät des Typs IBM 2520 (Lesen und Stanzen).
254Ø-1	Irgendein Lochkartengerät des Typs IBM 2540 (Lesen).
254Ø-2	Irgendein Lochkartengerät des Typs IBM 2540 (Stanzen in gleiche Karte).
35Ø5	Irgendein Kartenleser des Typs IBM 3505.
3525	Irgendein Kartenstanzer des Typs IBM 3525.

3. Durch einen *Gruppennamen*, Darunter versteht man symbolische Namen, die bei der Systemgenerierung vergeben werden. Gewöhnlich werden folgende Namen verwendet:

SYSSQ	Irgendein Magnetbandgerät oder ein externes Speichergerät mit direkter Zugriffsmöglichkeit, das für die temporäre Speicherung konsekutiv organisierter Datenmengen zur Verfügung steht.
SYSDA	Irgendein Speichergerät mit direkter Zugriffsmöglichkeit, auf dem Platz für Arbeitsspeicher zur Verfügung steht.
SYSCP	Irgendein Lochkartenstanzgerät.
TAPE	Dieser Name ist im System nicht erforderlich, doch wird er vielfach benutzt, um Laufwerke für Magnetbänder zu spezifizieren, die temporären Speicherungszwecken dienen sollen.

Ein Gerät kann durchaus durch mehr als einen Gruppennamen angesprochen werden.

Der Systemprogrammierer kann dafür sorgen, daß bei der Systemgenerierung gewisse Scheinadressen für Geräte ins System aufgenommen werden. Die dadurch spezifizierten Scheingeräte gelten als virtuelle Geräte; sie sind dann ebenfalls dem System unter dem für sie vergebenen Namen bekannt. Virtuelle Geräte können zur Aufnahme temporärer konsekutiv organisierter Datenmengen vorgesehen werden. Die auf solche Datenmengen bezogenen EA-Operationen eines Programmes werden jedoch nicht vollendet. Stattdessen fängt der Supervisor die betreffende EA-Operation ab und schreibt den Datensatz in ein Gebiet des virtuellen Speichers bzw. holt den Datensatz von einem solchen ab. Da die durch den Seitenwechsel bedingten EA-Operationen schneller ablaufen als die regulären EA-Operationen, mündet ein solches Vorgehen immer in einer schnelleren Ausführung der EA, auch wenn diese Methode zu einem häufigeren Seitenwechsel führt. Man nennt diese Art von EA-Operationen virtuelle EA [1]. Die Kanal-Geräte-Scheinadressen können auch zu Gruppen zusammengefaßt werden, wodurch sie durch Gruppennamen ansprechbar sind. Es gibt hierfür keinen standardisierten Gruppennamen, meist fällt die Wahl auf VIO oder SYSVIO als passablen Namen.

10.13. Das JOB-Steuerstatement im Betriebssystem OS

Jeder Job, der unter Steuerung des Betriebssystems OS ausgeführt werden soll, muß mit einem JOB-Steuerstatement beginnen. Allgemein lautet dessen Form (Abb. 10.5.).

1) Im angloamerikanischen Sprachraum als Virtual Input/Output, abgekürzt VIO, bezeichnet

```
//jobname  JOB  [([abnr][,abinf])]
               [,programmierer]
               [ADDRSPC={VIRT}]
                        {REAL}
               [CLASS=jobklasse]
               [COND=((bed),...)]
               [MSGCLASS=x]
               [MSGLEVEL=(x,y)]
               [PERFORM=klasse]
               [PRTY=vorrang]
               [REGION=nK]
               [TIME=(min,sek)]
               [TYPERUN={HOLD}]
                        {SCAN}
```

Abb. 10.5. Das JOB-Steuerstatement im Betriebssystem OS

Alle JOB-Steuerstatements beginnen mit zwei Schrägstrichen, die auf den Spalten 1 und 2 erscheinen müssen. Der Jobname fängt auf der Spalte 3 an; er muß mit einem Buchstaben eingeleitet werden. Die drei Zeichen $, # und @ werden dabei als Buchstaben angesehen. Die restlichen Zeichen des Namens können Buchstaben oder Ziffern sein. Die gesamte Zeichenzahl des Jobnamens darf jedoch nicht größer als 8 sein. Auf den Namen können ein oder mehrere Leerzeichen folgen, eins muß stets vorhanden sein. An diese schließt sich die Operationsbezeichnung an, in unserem Falle also JOB. Auf die Operationsbezeichnung muß wiederum mindestens ein Leerzeichen folgen, es können freilich auch mehrere erscheinen. Danach sind die Operanden aufzuführen. Neben diesen sind auch Bemerkungen erlaubt; sie müssen allerdings hinter dem letzten Operanden stehen, von diesem durch ein oder mehrere Leerzeichen getrennt. Die Bemerkungen dürfen außerdem nicht über die Spalte 71 hinausreichen. Operanden, die ohne Gleichheitszeichen niederzuschreiben sind, gelten als Positionsoperanden. Sie müssen unbedingt in der vorgegebenen Reihenfolge aufgeführt werden. Alle Operanden, die in sich ein Gleichheitszeichen einschließen, werden Schlüsselwortoperanden genannt und können in einer beliebigen Reihenfolge notiert werden. Die Operanden werden untereinander durch Kommas getrennt. Sind Operanden so lang, daß sie nicht auf eine Lochkarte passen, so können sie über so viele zusätzliche Karten hinaus fortgesetzt werden, wie benötigt werden. Muß oder soll auf eine Karte eine Fortsetzungskarte folgen, so muß sie mit einem Komma, d.h. nach einem Operanden, abgeschlossen werden. Ein vom Leerzeichen verschiedenes Zeichen kann auf die Spalte 72 einer fortzusetzenden Karte gestellt werden, aber verpflichtend ist ein solches Vorgehen nicht, denn Fortsetzungskarten werden auf eine andere Weise erkannt. Sie müssen nämlich auf den Spalten 1 und 2 die beiden üblichen Schrägstriche enthalten, die Spalte 3 hat jedoch leer zu bleiben. Die weiteren Operanden können irgendwo auf einer der Spalten 4 bis einschließlich 16 starten.

Der Jobname muß unbedingt im Jobsteuerstatement erscheinen. Dadurch wird dem OS-System ein Name präsentiert, den es in Nachrichten benutzen kann, die an den Maschinenbediener hinausgeschickt werden. Die Abrechnungsnummer abnr und die Abrechnungsinformation abinf sowie der Operand programmierer sind Positionsoperanden, die aufgeführt werden können , aber nicht müssen, d.h. sie sind sogenannte Optionen. Der Systemprogrammierer kann jedoch das Betriebssystem OS durch einen Eingriff so modifizieren, daß einer dieser Operanden oder sogar beide zu verpflichtenden Angaben werden. Wir sollten hierzu vermerken, daß abnr und abinf als Teile eines umfassenden Operanden, den wir als Abrechnungsoperanden charakterisieren wollen, zu gelten haben, weswegen sie ja auch von einem Klammernpaar umgeben sind. Fehlt der Ab-

rechnungsoperand, so muß dem Operanden programmierer ein Komma vorausgehen, um die Auslassung des ersten Operanden zu signalisieren. Können beide Operanden weggelassen werden, so dürfen in diesem Fall die Kommas unterdrückt werden. Der Abrechnungsoperand nimmt normalerweise die o.a. Form

(abnr,abinf)

an. Ist nur die Angabe der Abrechnungsnummer abnr erforderlich, so darf das Klammernpaar weggelassen werden. Wenn entweder abnr oder abinf irgendein Sonderzeichen mit Ausnahme des Bindestriches (Minuszeichens) enthalten, so muß die betreffende Angabe zusätzlich von zwei einfachen Anführungszeichen eingeschlossen werden. Soll das Sonderzeichen & verwendet werden, so müssen zwei dieser Zeichen codiert werden; das gleiche gilt für die einfachen Anführungszeichen. Betrachten wir einige Beispiele:

```
//BEISP1  JOB  (3227,AUDI8ØL)
//BEISP2  JOB  35Ø
//BEISP3  JOB  (,'IFA BK 35Ø')
```

Beim dritten Beispiel ist zu beachten, daß das zu Beginn, d.h. nach der geöffneten Klammer stehende Komma das Fehlen des Teiloperanden abnr andeutet.

Durch den zweiten Operanden programmierer wird der Programmierer festgehalten; gewöhnlich wird hierbei der Name des Programmierers erscheinen. Doch kann eigentlich hier jede Information aufgeführt werden, sofern sie mindestens ein Zeichen, jedoch nicht mehr als 20 Zeichen umfaßt. Wird in diese Information ein vom Punkt verschiedenes Sonderzeichen aufgenommen, so muß der gesamte Operand in einfache Anführungszeichen eingeschlossen werden. Soll eines der Sonderzeichen & oder ' unter programmierer erscheinen, so sind an seiner Stelle deren zwei zu notieren. Doch verfolgen wir wiederum einige Beispiele:

```
//BEISP4  JOB  1954,C.B.G
//BEISP5  JOB  ,'O''NEIL'
//BEISP6  JOB  (8Ø31,BK),'A&&R'
```

Im 6. Beispiel ist unter programmierer die Information A&R angegeben.

Die restlichen Operanden sind sämtlich Schlüsselwortoperanden. Einige von ihnen gehören nur zur Variante VS1, andere nur zur Variante VS2 des Betriebssystems OS. Werden diese Operanden für eine Variante niedergeschrieben, für die sie nicht definiert sind, so werden sie ignoriert. Doch wenden wir uns nun den einzelnen Operanden und ihrer Bedeutung zu.

Durch den Schlüsselwortoperanden

```
ADDRSPC=VIRT
```

wird festgelegt, daß der Job im virtuellen Speicher ausgeführt werden soll,

```
ADDRSPC=REAL
```

besagt jedoch, daß für den Job Realspeicher zur Verfügung gestellt werden soll. Durch den Operanden REGION wird dessen Größe in nk Bytes, d.h. n Kilobytes festgelegt. Fehlt dieser Operand, so wird per Standard

```
ADDRSPC=VIRT
```

unterstellt. Wird diese Form niedergeschrieben oder unterstellt, so wird der Operand REGION bei der Variante VS1 ignoriert. Bei der Variante VS2 wird unter REGION der Maximalumfang des virtuellen Speichers genannt, der verfügbar gemacht werden soll, wenn das Benutzerprogramm zusätzlichen virtuellen Speicher anfordert.

Durch den Operanden CLASS wird die Jobklasse festgelegt. Fehlt dieser Operand, so wird

```
CLASS=A
```

angenommen.

Durch den Operanden MSGCLASS wird die Ausgabeschreiberklasse spezifiziert, die zum Drucken von Nachrichten benutzt werden soll, die bei der Ausführung dieses Jobs anfallen. Fehlt dieser Operand, so wird die Klasse A angenommen.

Durch den Operanden

```
ADDRSPC=VIRT
```

wird festgelegt, daß der Job im virtuellen Speicher ablaufen soll. Wird stattdessen REAL angegeben, so soll er im Realspeicher ausgeführt werden. Durch den Operanden REGION wird dabei der Umfang des Realspeichers spezifiziert, der dem Job zugeordnet werden soll. Wird der ADDRSPC-Operand weggelassen, so gilt

```
ADDRSPC=VIRT
```

als Standard. Wird diese Form niedergeschrieben oder per Standard angenommen, so wird der Operand REGION von der Variante VS1 des Betriebssystems ignoriert. Bei der Variante VS2 hingegen wird durch diesen Operanden der Maximalumfang des virtuellen Speichers festgelegt, der dann verfügbar gemacht wird, wenn das Benutzerprogramm zusätzlichen virtuellen Speicher anfordert.

Durch die Niederschrift des Operanden

```
MSGLEVEL=1
```

wird veranlaßt, daß alle Jobsteuerstatements des betreffenden Jobs aufgelistet werden.

Durch

```
MSGLEVEL=Ø
```

wird die Auflistung unterdrückt. Durch

```
MSGLEVEL=2
```

wird erreicht, daß alle Jobsteuerstatements aufgelistet werden, ausgenommen hiervon sind diejenigen, die von katalogisierten Prozeduren (siehe Abschnitt 10.16.) her stammen. Man kann die zum Schlüsselwort gehörende Angabe auch in der Form (x,y) niederschreiben. Dabei kann x die Werte 0,1,2 annehmen, wobei diese die gleiche Bedeutung besitzen, die wir soeben geschildert haben. Durch y=0 wird veranlaßt, daß keine Auflistungen, die die Zuordnung der Datenmengen zu EA-Geräten beschreiben, in den Augenblicken erfolgen, in denen die Jobsteps gestartet und beendet werden. Wird für y der Wert 1 niedergeschrieben, so werden solche Zuordnungen aufgelistet. Standardmäßig gilt für den Operanden MSGLEVEL die Angabe (Ø,1).

Enthält ein Job mehrere Steps, d.h. werden mehrere Programme ausgeführt, so kann die Ausführung eines Jobsteps von einem Bedingungsschlüssel abhängig gemacht werden. Jedes Programm gibt am Schluß seiner Ausführung einen Wert an diesen Schlüssel zurück. Normalerweise zeigt Ø die erfolgreiche Vollendung an, höhere Werte hingegen bedeuten ein wenig erfolgreichere Ausführung. So bedeutet z.B. der

Wert 16 bei einem Kompilierer, daß er nicht in der Lage war, ein Quellenprogramm zu übersetzen. Deshalb gibt es in einem solchen Fall auch keine Möglichkeit, in einem nachfolgenden Jobstep den übersetzten Code zu binden. Jede Bedingung bed ist in der Form

(wert,operator)

zu codieren. Unter wert kann eine vorzeichenlose ganze Zahl zwischen Ø bis einschließlich 4Ø95 aufgeführt werden. Als Operatoren können die folgenden dienen:

GT	größer als
GE	größer als oder gleich
EQ	gleich
NE	nicht gleich
LE	kleiner als oder gleich
LT	kleiner als

Wird eine der angegebenen Bedingungen von einem ausgeführten Programm erfüllt, so wird der Rest des Jobs getilgt. Somit bedeutet z.B. das Jobsteuerstatement

```
//BEISP7  JOB  COD=((16,GE),=8,EQ))
```

folgendes: Die restlichen Jobsteps gelangen nicht mehr zur Ausführung, wenn ein Jobstep einen Bedingungsschlüssel abliefert, der größer als oder gleich 16 bzw. gleich 8 ist. Wenn nur eine Bedingung niedergeschrieben zu werden braucht, so können die äußeren Klammern wegfallen. Im Beispiel 8 wird ein solcher Fall demonstriert:

```
//BEISP8  JOB  COND=(16,GE)
```

Maximal können acht Bedingungen fixiert werden.

Bei Betriebssystemen der Variante VS2 kann durch den Operanden PERFORM die Zuweisung eines Jobs zu einer bestimmten Leistungsklasse erfolgen. Als Wert für klasse kann eine natürliche Zahl zwischen 1 und 255 angegeben werden. Die Leistungsklassen selbst werden vom Systemprogrammierer definiert. Eine Klasse kann z.B. für Jobs benutzt werden, für die die Nutzbarmachung der EA minimiert werden soll, einige andere Klassen können vielleicht für Jobs dienen, für die die ZE-Zeiten [1] minimiert werden sollen.

Die VS1-Systeme und einige VS2-Systeme ermöglichen durch Zuhilfenahme des Operanden PRTY die Festlegung einer Abfertigungspriorität innerhalb der betreffenden Jobklassen. Bei der Auswahl des nächsten auszuführenden Jobs innerhalb einer Jobklasse kommt derjenige zum Zuge, der die höchste Priorität aufweist. Für vorrang können als Werte ganze Zahlen zwischen 0 bis einschließlich 13 angegeben werden. Der Wert 13 wird gewöhnlich für das Maschinenbedienungspersonal reserviert und sollte deshalb auch nicht vom Programmierer vergeben werden.

Durch den Operanden

TIME=(min,sek)

kann die Zeit begrenzt werden, die die Zentraleinheit für den gesamten Job aufwenden darf. Wird diese überschritten, so bricht das System automatisch die Jobausführung ab.

1) Benutzungszeiten der Zentraleinheit (ZE)

Die Klammern können entfallen, wenn die Sekundenangabe sek weggelassen und nur die Minuten unter min angegeben werden. Durch

```
TIME=144Ø
```

wird dem System angezeigt, daß der Job auf immer laufen kann, denn Werte größer als 1440 sind nicht erlaubt.

Wenden wir uns nun dem Operanden

```
TYPERUN=HOLD
```

zu. Durch ihn wird der so gekennzeichnete Job so lange in eine Haltewarteschlange gestellt, bis der Maschinenbediener einen Freigabebefehl (RELEASE) für diesen Job erteilt. Sicher wird dieser Operand nur dann angewendet werden, wenn ausdrücklich auf seine Spezifizierung gedrungen wird. Durch

```
TYPERUN=SCAN
```

wird bewirkt, daß die Steuerstatements hinsichtlich von Fehlern durchsucht werden; die Ausführung des betreffenden Jobs wird jedoch nicht eingeleitet. Bei einem solchen Vorgehen kristallisieren sich auf eine bequeme Art und Weise Fehler beim Gebrauch der Jobsteuersprache heraus, die man sonst erst mühselig suchen müßte. Es ist jedoch zu beachten, daß die Angabe SCAN von der Variante VS2 nicht unterstützt wird.

10.14. Das EXEC-Steuerstatement im Betriebssystem OS

Innerhalb eines Jobs werden die einzelnen Jobsteps durch EXEC-Steuerstatements definiert. Diese Statements stellen jeweils das erste Steuerstatement eines Jobsteps dar, d.h. die ihnen äquivalenten Steuerkarten liegen stets an der Spitze der Kartenmenge eines Jobsteps. Allgemein kann das EXEC-Statement die in Abb. 10.6. aufgeführte Form annehmen.

```
//stepname    EXEC  { [PROC=]prozedurname  }
                    { PGM=programmname     }
                    { PGM=*.stepname.ddname }
                    [ACCT=(abrinf)]
                    [ADDRSPC={VIRT}]
                             {REAL}
                    [COND=((bed),...))]
                    [DPRTY=(x,y)]
                    [PARM=wert]
                    [DYNAMNBR=n]
                    [PERFORM=klasse]
                    [REGION=nK]
                    [TIME=(min,sek)]
```

Abb. 10.6. Das Steuerstatement EXEC im Betriebssystem OS

Wird unter stepname der Name eines Steps niedergeschrieben, so wird er zur Identifizierung des Steps innerhalb des Jobs herangezogen. Auf den Jobstep wiederum kann man sich stets beziehen, indem man den Jobnamen, gefolgt von einem Punkt und dem Stepnamen, codiert. Es gibt eigentlich nur einen Grund, den Stepnamen vorzusehen: Ein nachfolgendes Jobsteuerstatement muß sich dieses Stepnamens bedienen. Wenn wir z.B. die Statementfolge

```
//BEISP9  JOB   ...
//STEP1   EXEC  PGM=IEWL
```

vor uns sehen, dann können wir uns auf den Jobstep über BEISP9.STEP1 beziehen.

Als erster Operand muß PROC oder PGM gebraucht werden, sogar dann, wenn ein Gleichheitszeichen erscheinen kann. Dieser Operand wird nämlich als Positionsoperand angesehen. Er identifiziert nämlich die auszuführende Aufgabe entweder als eine katalogisierte Prozedur (siehe dazu Abschnitt 10.16.) oder als Name eines Programmes in der Bibliothek SYS1.LINKLIB bei Benutzung von PGM= oder sogar durch eine Bezugnahme auf ein DD-Steuerstatement (siehe dazu Abschnitt 10.15.) eines vorangegangenen Jobsteps innerhalb des gleichen Jobs.

Mittels des ACCT-Operanden werden die Abrechnungsinformationen festgelegt, die für den Jobstep gelten. Wenn nur eine Angabe vorgenommen wird, sind die Klammern nicht erforderlich. Werden andere Sonderzeichen außer dem Bindestrich (Minuszeichen) aufgeführt, so muß der Parameter dieses Operanden in Anführungszeichen und nicht in Klammern eingeschlossen werden. Beispiele für diesen Operanden sind:

```
ACCT=(STEP13,TEST)
ACCT=STEP14
ACCT='STEP15,BK+41'
```

Die Operanden ADDRSPC, PERFORM und REGION können beim EXEC-Steuerstatement in der gleichen Weise wie beim JOB-Steuerstatement gebraucht werden. Werden sie an beiden Stellen niedergeschrieben, so wird der Parameter ignoriert, der beim EXEC-Statement steht, und nur derjenige benutzt, der im JOB-Statement erscheint. Der Operand TIME besitzt beim EXEC-Statement die gleiche Form wie beim JOB-Statement; selbstverständlich gilt das spezifizierte Zeitlimit für die ZE nur für den Jobstep. Diese Zeit geht mit in die Jobzeit ein. Fehlt die Niederschrift des TIME-Operanden, so wird eine Standardzeit zur Begrenzung der ZE-Inanspruchnahme eingesetzt; diese ist vom Systemprogrammierer zu definieren. Gewöhnlich wird hierfür 30 Minuten gewählt.

Die Bedingungen bed beim COND-Operanden können in einer der nachstehenden Formen festgelegt werden:

```
(schlüssel,operator,stepname)
(schlüssel,operator)
EVEN
ONLY
```

Unter stepname muß dabei der Stepname erscheinen, dessen Bedingungsschlüssel geprüft werden soll. Fehlt stepname , so werden alle vorhergehenden Steps des Jobs getestet. Der Parameter EVEN besagt, daß der betreffende Step sogar dann ausgeführt werden soll, wenn der Job bei einem früheren Step infolge einer anderen Prüfung des Bedingungsschlüssels abgebrochen wurde. Die Angabe von ONLY hat hingegen zur Folge, daß der betreffende Job durch einen früheren Test des Bedingungsschlüssels abgebrochen wurde. Selbstverständlich können EVEN und ONLY nicht gleichzeitig für denselben Jobstep festgelegt werden.

Der Operand PARM ermöglicht die Übergabe von maximal 100 Informationszeichen an das auszuführende Programm. Die Gesamtinformation ist in Anführungszeichen einzuschließen und die zu übergebenden Parameter sind durch Kommas zu trennen. Die Anführungszeichen können durch Klammern ersetzt werden, wenn keine Sonderzeichen benutzt werden. Die hier aufgeführten Werte dienen oft dazu, die vom Systemprogrammierer festgelegten Standardwerte außer Kraft zu setzen. Beispiele hierfür sind:

```
PARM=(LIST,CHAR6Ø,FLAGW)
PARM='XREF,X-C'
```

Ist es erforderlich, ein einzelnes Anführungszeichen oder &-Zeichen zu übergeben, so muß dieses zweifach codiert werden.

Der Operand DPRTY kann in der Variante VS2 des Betriebssystemes OS zur Festlegung einer Abfertigungspriorität für den Jobstep benutzt werden. Der hierfür herangezogene Wert ergibt sich aus dem um y vermehrten 16-fachen Wert von x. Sowohl für x als auch für y gilt der Wertebereich, der sich aus den ganzen Zahlen zwischen 0 bis einschließlich 15 zusammensetzt.

Der Operand DYNAMNBR wird in der Variante VS2 dazu verwendet, die Zahl der nicht in Benutzung befindlichen Datenmengen festzulegen, die dem betreffenden Jobstep zugeordnet werden müssen. Wird er weggelassen, so wird 0 als Wert angenommen. Die Gesamtzahl der Datenmengen, auf die sich das auszuführende Programm bezieht, darf die Summe aus n und der Anzahl der für diesen Jobstep spezifizierten DD-Steuerstatements nicht überschreiten.

10.15. Das DD-Steuerstatement im Betriebssystem OS

Jede Datenmenge, auf die sich ein Programm bezieht, muß dem System durch ein DD-Steuerstatement [1] bekannt gemacht werden. Seine allgemeine Form lautet:

```
//ddname  DD  operanden
```

Der Anhang L enthält eine Übersicht über alle möglichen Operanden.

Unter ddname wird der Name verstanden, der der Datenmenge im Programm gegeben wird, d.h. er stellt den Dateinamen dar. Folgende Namen dürfen hierfür nicht benutzt werden:

```
JOBCAT
JOBLIB
STEPCAT
STEPLIB
SYSABEND
SYSUDUMP
SYSCHK
```

Ein Positionsoperand ist erlaubt. Er kann wie folgt heißen:

a) *
Durch diesen Operanden wird angezeigt, daß die Datenmenge im Eingabestrom liegt und auf dieses Steuerstatement folgt. Sie wird beendet durch die Zeichenfolge // oder /* auf den Spalten 1 und 2 eines Steuerstatements.

b) DATA
Durch diesen Operanden wird angezeigt, daß die Datenmenge auf dieses Steuerstatement im Eingabestrom folgt und durch das erste Statement abgeschlossen wird, das die Zeichenfolge /* auf den Spalten 1 und 2 aufweist.

c) DUMMY
Durch diesen Operanden wird angezeigt, daß keine aktuellen EA-Operationen stattfinden sollen. Bei einer Eingabedatenmenge verursacht die erste Leseoperation,

1) DD ist die Abkürzung von „data definition" (Datendefinition)

daß die Bedingung „Dateiende" entsteht und dem Programm überstellt wird. Bei Ausgabedatenmengen werden alle Schreiboperationen als erfolgreich abgeschlossen betrachtet. Dieser Operand kann sicher sehr nützlich beim Austesten von Programmen sein.

Wird als Operand * oder DATA niedergeschrieben, so können durch den weiteren Operanden DLM=xx irgendwelche zwei Zeichen unter xx erscheinen, die dann an Stelle von /* zur Markierung des Datenmengenendes dienen können.

Für eine über SYSOUT auszugebende Datenmenge ist zu codieren:

```
SYSOUT=(klasse,programm,formnr)
```

Unter klasse wird die Ausgabeklasse festgelegt, unter programm der Name eines speziellen Ausgabeschreibers, der für die betreffende Datenmenge benutzt werden soll, und unter formnr kann schließlich eine vierstellige Formularnummer erscheinen. Wird formnr nicht angegeben, so werden die Ausgabezeilen der Datenmenge auf diejenigen Formulare gedruckt, die gerade im Drucker eingespannt sind. Fehlt der zweite Parameter, so wird der von der IBM offerierte Standardausgabeschreiber herangezogen. Kommas, die auf den letzten Parameter folgen, können weggelassen werden. Wird nur der erste Parameter, d.h. klasse, benutzt, so können auch die Klammern weggelassen werden. Beispiele hierfür sind:

```
DSNAME=dsname
DSN=dsname(gliedname)
DSN=*.ddname
DSN=*.stepname.ddname
DSNAME=&&dsname
```

Bei einem Zeilendrucker des Typs IBM 3211 kann außerdem der FCB-Operand in einer der folgenden Formen aufgeführt werden:

```
FCB=(bild,ALIGN)
FCB=(bild,VERIFY)
FCB=bild
```

Unter bild wird der Code verstanden, der als Abbild für einen Formularsteuerpuffer in die Bibliothek SYS1.IMAGELIB aufgenommen wurde. Durch ALIGN wird eine Prüfung seitens der Maschinenbedienung hinsichtlich der korrekten Formularausrichtung vor Beginn des Druckens verlangt. Die Angabe des Parameters VERIFY hat schließlich zur Folge, daß nicht nur die Formularausrichtung, sondern auch das FCB-Abbild von der Maschinenbedienung überprüft werden muß. Wird der FCB-Operand weggelassen (gewöhnlich geschieht dies auch), so wird ein standardisiertes FCB-Abbild benutzt. Bei einem mit dem Universaltypensatz ausgerüsteten Zeilendrucker können wir uns des UCS-Operanden in einer der folgenden Formen bedienen:

```
UCS=(tsc,FOLD,VERIFY)
UCS=(tsc,FOLD)
UCS=(tsc,,VERIFY)
```

Der Typensatzcode tsc wird in der Bibliothek SYS1.IMAGELIB gefunden. Durch Angabe von VERIFY wird verlangt, daß die Maschinenbedienung überprüft, ob die passende Druckkette bzw. die passenden Druckglieder montiert sind. Durch Angabe des zweiten Parameters FOLD wird bewirkt, daß die Kleinbuchstaben in die Großbuchstaben übergeführt werden, d.h. sinnbildlich eine Umschaltung erfolgt.

Für eine beliebige Ausgabedatenmenge, meistens jedoch für Datenmengen, die SYSOUT zugeordnet werden, können wir den Operanden

OUTLIM=n

spezifizieren. Unter n ist dabei die Maximalzahl von Sätzen zu verstehen, die geschrieben werden sollen. Wird der n zugewiesene Wert überschritten, so wird der Job vom Betriebssystem aufgehoben. Bei Zuweisung von Ø zu n wird kein Limit auferlegt. Fehlt dieser Operand, so tritt an die Stelle von n ein Standardwert für das Limit, der vom Systemprogrammierer festzulegen ist.

Wenn die Datenmenge nicht in den Systemkatalog aufgenommen werden soll, so haben wir den folgenden Operanden in einer der beiden nachstehenden Formen zu verwenden:

UNIT=einheit
UNIT=AFF=ddname

Mit diesem Operanden verweisen wir auf die Einheit, auf die die Datenmenge plaziert werden soll. Die Einheit kann in irgendeiner der Weisen spezifiziert werden, die wir im Abschnitt 10.12. kennengelernt haben. Die Angabe von AFF=ddname bedeutet, daß die Datenmenge der gleichen Kanalgeräteadresse zugewiesen werden soll wie diejenige Datenmenge, die durch das DD-Steuerstatement gekennzeichnet ist, das den Namen ddname aufweist und die zuvor im gleichen Jobstep verarbeitet wurde. Damit kann die Zahl der benötigten EA-Geräte reduziert werden, wenn zwei Datenmengen nicht zur gleichen Zeit verarbeitet werden. Wir müssen ebenfalls die einzelnen zu benutzenden Datenträger angeben. Hierfür setzen wir den VOL-Operanden in einer der nachstehenden Formen ein:

VOL=SER=snr
VOLUME=SER=(snr[,snr]···)
VOL=(,,,dz)

Die erste Form wird benutzt, wenn nur ein Datenträger mit einer spezifischen Reihennummer (Seriennummer) snr eingesetzt wird, die zweite bei Einsatz von mehr als einem Datenträger. Die letzte wird für Ausgabedatenmengen verwendet, wenn die Anzahl der Datenträger größer als 1 werden kann. Unter dz ist dann die mutmaßliche Datenträgerzahl aufzuführen. Für diesen Operanden kann sowohl das Schlüsselwort VOLUME als auch dessen Abkürzung VOL benutzt werden, beide Formen gelten als äquivalent. Bei temporären Datenmengen kann VOL weggelassen werden, das Betriebssystem wählt dann selbst geeignete Arbeitsdatenträger für die Aufnahme der Datenmenge aus.

Bei einer neu zu erstellenden Ausgabedatenmenge müssen wir den bereitzustellenden Platz angeben, wenn die Datenmenge auf eine SEDZ [1] untergebracht werden soll. Ist der Platzbedarf spezifiziert, wird jedoch die Datenmenge nicht auf eine SEDZ gelenkt, so bleibt die Platzbedarfsangabe unbeachtet. Der Platzbedarf kann durch den SPACE-Operanden in einer der folgenden Formen genannt werden:

SPACE=(untert,(primz,zweita,verz),RLSE)
SPACE=(untert,(sekz,zweita,verz)

Wird der Unterparameter RLSE angegeben, so wird der unbenutzte Platz am Ende des zugeordneten Platzes an das Betriebssystem zurückgegeben, wenn die Verarbeitung abgeschlossen ist. Unter untert muß die Unterteilungsart des Speicherplatzes angegeben werden; dabei ist TRK zu benutzen, wenn der Platzbedarf auf Spuren, CYL dagegen, wenn er auf Zylinder bezogen werden soll. Bei Angabe einer Zahl wird diese als Blocklänge angesehen. Unter primz hat die Zahl der Spuren, Zylinder bzw. Blöcke zu erscheinen, die zuerst (primäre Zuordnung) für die Datenmenge bereitgestellt werden sollen. Reicht dieser Platz nicht aus, so können bis zu 15 weiteren Platzzuordnungen

1) SEDZ ist die Abkürzung für Speichereinheit mit direkter Zugriffsmöglichkeit

(sekundäre Zuordnung) durch sekz erfolgen. Der Unterparameter verz spielt bei einer untergliederten Datenmenge eine Rolle. Für ihn ist die Anzahl der Blöcke aufzuführen, die zur Aufnahme des Verzeichnisses der Glieder der untergliederten Datenmenge erforderlich sind. Bei einer indexsequentiell organisierten Datenmenge wird unter verz die Anzahl der für das Inhaltsverzeichnis benötigten Zylinder festgelegt.

Eine nichttemporäre Datenmenge muß benannt werden. Hierzu dient der Operand mit dem Schlüsselwort DSNAME bzw. dessen Abkürzung DSN, mit dessen Hilfe der Datenmengenname festgelegt wird. Er kann in einer der folgenden Formen erscheinen:

```
SYSOUT=A
SYSOUT=(A,,1296)
SYSOUT=(E,AUSGABE,1297)
SYSOUT=(B,AUSGABE)
```

Die erste Form wird am meisten benutzt. Beim Ansprechen eines Gliedes einer untergliederten Datenmenge bezieht sich dsname auf den Datenmengennamen und gliedname auf den Namen des zu verarbeitenden Gliedes. Ein Sternzeichen bedeutet, daß es sich um die gleiche Datenmenge handelt, die durch ein vorhergehendes DD-Steuerstatement im gleichen Jobstep oder durch ein DD-Steuerstatement in einem vorhergehenden Jobstep beschrieben sind. Zwei aufeinanderfolgende &-Zeichen zeigen schließlich an, daß es sich um eine temporäre Datenmenge handelt, die nach Erreichen des Jobendes vom System wieder zerstört werden kann.

Oftmals werden auch zwei weitere Operanden benötigt, nämlich

```
DISP=(pm1,pm2,pm3)
LABEL=(lfnr,typ)
```

Besprechen wir zunächst den DISP-Operanden, durch den über die Datenmenge disponiert werden kann. Sein erster Parameter pm1 kann NEW, OLD, SHR oder MOD lauten. NEW bedeutet, daß eine neue Datenmenge angelegt werden soll, OLD hingegen verweist auf eine bereits existierende. SHR zeigt ebenfalls an, daß die Datenmenge bereits vorhanden ist, jedoch außerdem zusätzlich, daß nichts dagegen einzuwenden ist, wenn außer diesem Jobstep auch ein anderer Job die betreffende Datenmenge mit benutzt (gemeinsame Benutzung einer Datenmenge durch mehrere Jobs). Durch MOD wird schließlich gesagt, daß eine existierende Datenmenge erweitert werden soll, indem neue Sätze an ihrem Ende hinzugefügt werden. Der zweite Parameter pm2 kann zu DELETE, KEEP, PASS, CATLG oder UNCATLG formuliert werden. DELETE hat zur Folge, daß die Datenmenge am Ende des Jobsteps zerstört werden kann, KEEP, daß sie über den Jobstep hinaus beibehalten werden soll. PASS wiederum bedeutet, daß die Datenmenge nach Erreichen des Jobstependes weiter existieren, jedoch am Jobende der Zerstörung unterliegen soll. CATLG besagt, daß neben der Beibehaltung (KEEP) auch Einheit, Datenträger und Datenmengenname in den Systemkatalog aufgenommen werden soll; UNCATLG hat Beibehaltung (KEEP) und zusätzlich die Entfernung der Katalogeintragung zur Folge. Der letzte Parameter pm3 kann DELETE, KEEP, CATLG oder UNCATLG lauten und wird an Stelle des zweiten Unterparameters benutzt, wenn der Jobstep abnormal zu Ende geht. Fehlt der dritte Parameter, so wird angenommen, daß er gleich dem zweiten ist. Wird der zweite Parameter weggelassen, so schließt NEW auch DELETE ein, OLD, SHR oder MOD zusätzlich auch KEEP. Wird nur der erste Parameter niedergeschrieben, so können die Klammern entfallen. Einige Beispiele sollen zur Illustrierung des Gebrauchs dieses Operanden dienen:

```
DISP=OLD
DISP=(NEW,KEEP,DELETE)
DISP=(,PASS)
```

Fehlt der erste Parameter, so wird NEW angenommen. Bei Weglassung des gesamten Dispositionsoperanden wird per Standard

```
DISP=(NEW,DELETE)
```

unterstellt.

Beim LABEL-Operanden (Kennsatzoperanden) wird durch den ersten Parameter die laufende Nummer der Datenmenge spezifiziert; fehlt er, so wird 1 angenommen und das Komma muß zur Kennzeichnung seines Fehlens niedergeschrieben werden. Durch den zweiten Parameter werden die Kennsatztypen (Etiketttypen) in folgender Weise umrissen:

SL	Standardkennsätze
SUL	Standard- und Benutzerkennsätze
NSL	Kennsätze, jedoch keine Standardkennsätze
NL	Keine Kennsätze (infolgedessen auch keine Bandmarke vor der ersten Datenmenge)
BLP	Übergehen der Kennsatzverarbeitung
LTM	Führende (vorausgehende) Bandmarke

Bei Weglassung des zweiten Parameters wird SL angenommen. Fehlt der LABEL-Operand gänzlich, so wird

```
LABEL=(1,SL)
```

als Standard unterstellt. Die Angabe von LTM bedeutet, daß ein Magnetband ohne Kennsätze vorliegt, bei dem jedoch der ersten Datenmenge auf dem Band eine Bandmarke vorausgeht. BLP besagt, daß das System alle Prüfungen von Kennsätzen unterlassen soll; einige Systeme betrachten BLP deshalb als NL.

Der Operand

```
DCB=(attr[,attr]···)
```

und der Operand

```
AMP=(attr[,attr]···)
```

werden benutzt, um die Informationen über die charakteristischen Eigenschaften der Datenmenge durch Attribute attr zu vervollständigen, wenn nämlich auf ihre Angabe im Programm verzichtet worden ist. Der Operand AMP wird hierbei für Datenmengen eingesetzt, die nach VSAM organisiert sind, der Operand DCB für alle anderen. Einige häufig benutzte DCB-Parameter sind:

DSORG	zur Festlegung der Verarbeitungsform,
RECFM	zur Festlegung der Satzart,
BLKSIZE	zur Festlegung der Blocklänge,
LRECL	zur Festlegung der Satzlänge,
OPTCD	zur Festlegung spezieller Optionen.

Sollen DCB-Informationen aus einem vorhergehenden DD-Steuerstatement kopiert werden, so lautet die Form des DCB-Operanden wie folgt:

```
DCB=(*.ddname[,attr]···)
DCB=dsname
DCB=*.stepname.ddname
```

Werden hierbei Attribute unter attr aufgeführt, so setzen sie diejenigen außer Kraft, die aus einem vorhergehenden DD-Steuerstatement übernommen werden. Wird durch

dsname ein Datenmengenname angegeben, so muß es sich um den Namen einer katalogisierten Datenmenge handeln, die auf einer SEDZ residiert, die vor Aufnahme der Ausführung des Jobsteps bereits montiert ist.

Bereits dieser kurze Streifzug hat gezeigt, daß die Regeln zur Abfassung der DD-Steuerstatements recht verwickelt und kompliziert sind. An dieser Stelle wollen wir uns deshalb vor Augen halten, daß meist weniger als 1 % aller im aktuellen täglichen Gebrauch benutzten DD-Statements mehr als nur einige wenige Basisoperanden einschließen.

10.16. Katalogisierte Prozeduren im Betriebssystem OS

Da gewisse Kombinationen von Steuerstatements, wie z.B. solche, die für das Übersetzen, Binden und Ausführen von Programmen benötigt werden, häufig auftreten, besitzt das Betriebssystem OS eine Prozedurbibliothek. Die Eintragungen in dieser Bibliothek sind nichts anderes als Mengen von Standardsteuerstatements. Sie können die Beschreibung eines Jobsteps oder einer Reihe von Jobsteps beinhalten. Damit ist gesagt, daß sie aus geeigneten EXEC- und DD-Steuerstatements bestehen. Besitzt das EXEC-Statement eine der Formen

```
//[stepname]  EXEC  PROC=prozedurname,...
//[stepname]  EXEC  prozedurname,...
```

so wird die Prozedur aus der Bibliothek mit dem Namen SYS1.PROCLIB herausgeholt und die zugehörigen Jobsteuerstatements aus der Bibliothek werden an Stelle des EXEC-Statements in den über SYSIN einlaufenden Eingabestrom eingefügt. Die Abb. 10.7. zeigt als Muster die von der IBM unterstützte Prozedur PLIXCL, die für das Übersetzen und das Binden, jedoch nicht für die Ausführung eines in PL/1 geschriebenen Programmes herangezogen werden kann.

```
PLIXCL   PROC  LKLBDSN='SYS1.PLIBASE'
PLI      EXEC  PGM=IELOAA,PARM='OBJECT,NODECK',REGION=1ØØK
SYSPRINT DD    SYSOUT=A,DCB=(RECFM=VBA,LRECL=125,BLKSIZE=629)
SYSLIN   DD    DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
               SPACE=(8Ø,(25Ø,1ØØ))
SYSUT1   DD    DSN=&&SYSUT1,UNIT=SYSDA,SPACE=(1Ø24,(6Ø,6Ø),,CONTIG),
               DCB=BLKSIZE=1Ø24
'LKED    EXEC  PGM=IEWL,PARM='XREF,LIST',COND=(9,LT,PLI),REGION=1ØØK
'SYSLIB  DD    DSN=&LKLBDSN,DISP=SHR
'        DD    DSN=SYS1.PLIBASE,DISP=SHR
/SYSLMOD DD    DSN=&&GOSET(GO),DISP=(MOD,PASS),UNIT=SYSDA,
/              SPACE=(1Ø24,(5Ø,2Ø,1),RLSE)
/SYSUT1  DD    DSN=&&SYSUT1,UNIT=SYSDA,SPACE=(1Ø24,(2ØØ,2Ø)),
/              DCB=BLKSIZE=1Ø24
/SYSPRINT DD   SYSOUT=A
/SYSLIN  DD    DSN=&&LOADSET,DISP=(OLD,DELETE)
/        DD    DDNAME=SYSIN
```

Abb. 10.7. Beispiel für eine Prozedur

Prozedurname	Ausgeführte Programme			
	Übersetzer	Lader	Binder	Ausführung des übersetzten Progr.
PLIXC	PL/1			
PLIXCG	PL/1	X		X
PLIXCL	PL/1		X	
PLIXCLG	PL/1		X	X
PLIXG				X
PLIXLG			X	X
FORTGC	FORTRAN			
FORTGCL	FORTRAN		X	
FORTGCLD	FORTRAN	X		X
FORTGCLC	FORTRAN		X	X
FORTGLC			X	X
COBUC	COBOL			
COBUCG	COBOL	X		X
COBUCL	COBOL		X	
COBUCLG	COBOL		X	X
COBULG	COBOL		X	X
ASMFC	Assembler			
ASMFCG	Assembler	X		X
ASMFCL	Assembler		X	
ASMFCLG	Assembler		X	X
LKED			X	
LKEDG			X	X

Abb. 10.8. Übersicht über die von der IBM unterstützten Prozeduren

Die in Abb. 10.7. vorgestellte Prozedur setzt sich aus zwei Prozedurschritten (Prozedursteps) zusammen, nämlich aus dem Step PLI und aus dem Step LKED. Der erste Step bewirkt die Übersetzung, der zweite das Binden. Zur Herstellung von Rückbeziehungen auf Statements innerhalb der Prozedur müssen wir die Form stepname.prozedurstepname benutzen, d.h. den Namen des EXEC-Statements, der die Prozedur aufruft, über einen Punkt mit dem Namen des EXEC-Statements innerhalb der Prozedur verknüpfen. Mittels des die Prozedur aufrufenden EXEC-Statements sind wir in der Lage, Operanden innerhalb der Prozedur zu ändern oder außer Kraft zu setzen. Diese Möglichkeit wollen wir am nachfolgenden Beispiel kennenlernen:

```
//SS EXEC PLIXCL,PARM.PLI='OBJECT,NODECK,LIST'
```

Dieses Statement ersetzt den gesamten PARM-Operanden, der in der Prozedur für deren Step PLI festgelegt wurde. Wenn eine Prozedur benutzt wird, so können die in ihr enthaltenen DD-Statements geändert werden, indem wir ein neues DD-Statement in der Form

```
//prozedurname.ddname  DD  operanden
```

niederschreiben. Unter operanden sind dabei die geänderten oder die neuen Operanden niederzuschreiben. Die geänderten DD-Statements müssen im Quelleneingabestrom in der gleichen Reihenfolge erscheinen wie ihre Originale in der Prozedur stehen. Um beispielsweise Änderungen bei den Dateien SYSUT1 und SYSPRINT im Prozedurstep LKED vorzunehmen, müssen wir die nachstehende Statementfolge einsetzen:

```
//SS   EXEC  PLIXCL
//LKED.SYSUT1   DD  SPACE=(1Ø24,(3ØØ,2Ø))
//LKED.SYSPRINT  DD  SYSOUT=C
```

Um neue DD-Steuerstatements einer Prozedur hinzuzufügen, müssen diese hinter alle modifizierten DD-Statements plaziert werden. DD-Steuerstatements mit einem der Operanden DATA oder * können nicht in Prozeduren aufgenommen werden. Standardisierte IBM-Prozeduren verwenden in der Regel SYSIN als ddname (Dateinamen) für solche Statements.

Die Abb. 10.8. vermittelt uns eine Übersicht über die Namen der von der IBM unterstützten Prozeduren.

Den Lader haben wir bisher noch nicht erwähnt. Das sich dahinter verbergende Programm ähnelt dem Binder (Linkage Editor), doch stellt es den Lademodul in den Speicher und nicht in eine Bibliothek. Außerdem vermissen wir bei ihm einige Einrichtungen des Binders. Er wird manchmal an Stelle des Binders für einfache Programme benutzt, weil er für eine schnellere Ausführung als der Binder sorgt.

Ein typisches Steuerkartenpaket zur Verwendung von PLIXCLG sieht beispielsweise so aus, wie es die Abb. 10.9. zeigt.

```
//JOBA  JOB  MSGLEVEL=1
//STEPA  EXEC  PLIXCLG
//PLI.SYSIN  DD  *
  (Quellenprogramm)
/*
//GO.DN  DD  DISP=NEW,UNIT=SYSDA,SPACE=(CYL,(1Ø,2))
//GO.DK  DD  SYSOUT=B
//GO.SYSIN  DD  *
  (Daten für das Quellenprogramm)
/*
```

Abb. 10.9. Beispiel für die Verwendung einer von der IBM unterstützten Prozedur

Gehen wir auf einige Details der Abb. 10.9. ein. Bei DN, DK und SYSIN handelt es sich um die Namen von Dateien, die im Benutzerprogramm angesprochen werden. Der Prozedurstepname GO wird stets für die Ausführungssteps, der Name LKED für die Bindersteps in den von der IBM unterstützten Prozeduren benutzt.

Aufgabengruppe 24

1) Klassifiziere die nachfolgenden Einheitennamen im Betriebssystem OS nach folgenden Gesichtspunkten:
   (1) Kanalgeräteadresse
   (2) Gerätetyp

(3) Gruppenname
(4) ungültiger Name
a) 25Ø1, b) 2A3, c) 25Ø1A3, d) SYSCP

2) Jedes der folgenden DD-Steuerstatements ist gültig. Nach Ausführung eines dieses Statement benutzenden Programmes soll jede der Dateien klassifiziert werden, wobei folgende Aspekte heranzuziehen sind:

(1) katalogisierte Datenmenge
(2) beibehaltene, jedoch nicht katalogisierte Datenmenge
(3) zerstörte Datenmenge

a) `//DD1 DD DISP=OLD,UNIT=333Ø,VOL=SER=333333,DSN=SYS2.DD1`
b) `//DD2 DD UNIT=SYSDA,DSN=&&KEEP`
c) `//DD3 DD DISP=OLD,DSN=SYS2.DD3`
d)
```
//DD4 DD DISP=(,CATLG),DSN=SYS2.DD4,UNIT=333Ø,VOL=SER=333333,
//         SPACE=(CYL,(3))
```
e) `//DD5 DD DISP=(OLD,UNCATLG),DSNAME=SYS5`

3) In welcher Bibliothek des Betriebssystems OS kann man erwarten, folgende Programme bzw. Prozeduren zu finden?

a) die katalogisierte Prozedur PLIXC
b) den PL/1-Kompilierer
c) die Subroutine für PL/1-Programme, die der Umwandlung von Zeichenketten in gepackte Dezimalzahlen dient
d) ein Programm zur Erstellung von Lohnlisten, das wöchentlich in einem Unternehmen abläuft
e) eine Pufferfüllung für die QNC-Druckeinrichtung des Zeilendruckers IBM 1403-N1.

4) Falls der erste Jobstep eines Programmes für den Bedingungsschlüssel den Wert 8 zurückgibt, so ist zu untersuchen, ob der zweite Jobstep ausgeführt wird. Im EXEC-Steuerstatement des zweiten Jobsteps ist dabei der folgende Operand enthalten:

a) `COND=(16,LT)`
b) `COND=(16,GT)`
c) `COND=((8,EQ),ONLY)`
d) `COND=((9,NE),(8,EQ))`
e) `COND=EVEN`
f) `COND=(8,LE)`

5) In der Variante VS1 des Betriebssystemes OS sind in die Jobwarteschlange Jobs in der nachstehenden Reihenfolge hineingestellt worden:

JOB1	CLASS=A	PRTY=2
JOB2	CLASS=B	PRTY=3
JOB3	CLASS=C	PRTY=4
JOB4	CLASS=D	PRTY=5
JOB5	CLASS=A	PRTY=3
JOB6	CLASS=C	PRTY=4

Gib an, in welcher Reihenfolge die Jobs initialisiert werden, wenn eine Partition vorliegt, die die Jobklassen DBCA verarbeitet!

6) Ordne die Eintragungen in der zweiten Spalte so um, daß sie zu den entsprechenden Eintragungen der ersten Spalte passen!

(a) JOB-Steuerstatement	(1) Kennzeichnung für den Beginn eines Jobsteps
(b) EXEC-Steuerstatement	(2) Erforderliches Statement für jede Datenmenge
(c) DD-Steuerstatement	(3) Kennzeichnung für den Beginn eines Jobs
(d) /*-Steuerstatement	(4) Kennzeichnung für das Ende einer Datenmenge

10.17. Die konsekutive (starr fortlaufende) Organisationsform für Datenmengen

Zu konsekutiv organisierten Datenmengen kann nur sequentiell, d.h. fortlaufend, zugegriffen werden. Die sequentielle Zugriffsmethode kann dabei auf zwei verschiedene Weisen verwirklicht werden. Eine Art besteht darin, daß im Programm ein Befehl zum Starten der EA-Operation hinausgeschickt wird und im Anschluß daran solange ein Warten veranlaßt wird, bis die Operation abgeschlossen ist. Erst dann wird mit der weiteren Verarbeitung fortgefahren. Wenn mehr als ein EA-Gebiet oder wenn ein Arbeitsgebiet benutzt wird, sorgen die EA-Routinen automatisch für eine Anforderung des nächsten Satzes, wodurch es gewöhnlich nicht mehr zu einem Wartezustand im Programm zu kommen braucht. Diese Art des Zugriffs wird als gepufferter oder erweiterter Zugriff bezeichnet. Bei COBOL ist es sogar die einzige Standardart der sequentiellen Zugriffsmethode.

Die andere Art zeichnet sich dadurch aus, daß zunächst ein Befehl emittiert wird, der die EA-Operation startet; im Anschluß daran wird die Verarbeitung fortgesetzt. Erst an einer späteren Programmstelle erfolgt die Aussendung eines SVC-Befehls, durch den ein Warten bis zur Vollendung der EA-Operation veranlaßt wird. Es wird nur ein EA-Gebiet benutzt, kein Arbeitsgebiet. Die EA-Routinen bewirken von sich aus niemals die Einleitung der nächsten EA-Operation im voraus. Diese Zugriffsart ist für Magnetbandgeräte und für SEDZ erlaubt; das Betriebssystem OS gestattet ihre Anwendung sogar für alle Gerätetypen. Bei Magnetbändern verwalten die EA-Routinen einen Zähler, in dem die Anzahl der Blöcke festgehalten wird, die bei einer Datenmenge seit Beginn der Verarbeitung derselben bereits angesprochen wurden; bei Plattenspeichern merken sich die EA-Routinen die Position des letzten gelesenen bzw. geschriebenen Satzes. Auf diese Informationen kann das Benutzerprogramm zurückgreifen; man spricht in einem solchen Fall von einer Merk- oder Notiz-Operation. Normalerweise wird sich die nächste EA-Operation auf den anschließenden Satz beziehen, doch kann im Programm auch eine Hinweisoperation aufgesetzt werden, um damit die Position des nächsten zu verarbeitenden Blockes, d.h. Satzes zu umreißen. Bei einem solchen Vorgehen sorgen die EA-Routinen automatisch für die Einstellung des Bandes bzw. des Plattenspeichers auf den gewünschten Platz. Diese Art des Zugriffs wird als ungepufferter Zugriff oder als Basiszugriff bezeichnet. Dateien, auf die diese Zugriffsart angewendet wird, werden mitunter Arbeitsdateien genannt, denn sowohl die Eingabe- als auch die Ausgabeoperation können mit Hilfe der Merk- bzw. Hinweisoperationen auf die gleiche Datei bezogen werden, bei Bedarf auch miteinander vermischt. Das Betriebssystem DOS unterstützt für diese Zugriffsart die Standardkennsätze (Standardetiketten) bei Bändern nicht. Alle Sätze müssen ungeblockt vorliegen. Das Betriebssystem DOS untersützt dabei nur die Satzarten F und U, das Betriebssystem OS zusätzlich auch die Satzart V. Auf diese Zugriffsart kann man sich in der AL [1] voll stützen; sie steht auch für PL/1 zur Verfügung, allerdings nur mit gewissen Einschränkungen. So können Eingabeoperationen nicht mit Ausgabeoperationen vermengt werden, außerdem sind Notiz- und Hinweisoperationen nicht durchführbar.

Bei in FORTRAN verfaßten Programmen wird die erweiterte Zugriffsart bei allen regulären EA-Operationen benutzt. Die Basiszugriffsart steht bei Plattenspeichern ebenfalls zur Verfügung, allerdings ohne Notiz- und Hinweisoperationen. Wird die erweiterte Zugriffsart eingesetzt, so kann im Betriebssystem DOS ein in FORTRAN geschriebenes Programm nur mit geblockten Sätzen arbeiten; das gilt sowohl für Magnetbänder als auch für Plattenspeicher.

1) AL ist die Abkürzung von „Assembler Language"

10.18. Die wahlfreie (direkte) Organisationsform für Datenmengen

Die wahlfreie Organisationsform kann nur für Datenmengen realisiert werden, die auf SEDZ residieren. Bei einer solchen Speichereinheit kann der direkte Zugriff zu einem Satz in folgender Weise praktiziert werden: Die Spur wird in irgendeiner Form angegeben; der individuelle Satz wird auf dieser dann dadurch eruiert, daß sie entweder nach dem spezifizierten Schlüssel durchsucht oder auf die angegebene Satznummer eingestellt wird. Ist vorgesehen, den Satz durch seinen Schlüssel zu lokalisieren, so kann wahlweise auch ein erweitertes Durchsuchen veranlaßt werden. Unter dem erweiterten Durchsuchen versteht man das folgende Vorgehen: Zunächst wird die eingestellte Spur nach einem Satz mit dem angegebenen Schlüssel durchsucht. Wird kein Satz gefunden, so wird das Durchsuchen auf der anschließenden Spur fortgesetzt, d.h. auf der Spur mit der nächsthöheren Spurnummer. Dieses Abtasten wird über soviele Spuren hinweg fortgeführt, wie es vom Programmierer vorgesehen wird, unter Umständen kann er sogar alle Spuren der gesamten Datenmenge durchsuchen lassen. Da das Durchsuchen nur auf den Einzelsatz bezogen wird, können die Sätze nicht geblockt werden. Beim Betriebssystem DOS liegt zusätzlich die Beschränkung auf die Satzarten F und U vor.

Spur- und Satznummer sind auf eine der folgenden vier Formatarten [1] anzugeben.

1) TTTR oder TTR [2]

   Eine auf drei oder zwei Bytes stehende ganze Dualzahl enthält die Spurnummer relativ zum Anfang der Datenmenge, wobei die Anfangsspur als Spur 0 gilt. Im Anschluß an die Spurnummer folgt die ein Byte umfassende Satznummer als ganzzahlige Dualzahl. Durch die Satznummer werden nur die auf der angegebenen Spur vorhandenen Sätze erfaßt.

2) TTTTTTTTRR

   Die relative Spurnummer wird durch eine acht Bytes lange ganze Dezimalzahl gekennzeichnet; die einzelnen Ziffern dieser Dezimalzahl sind mit Zonen versehen. Im Anschluß daran folgt die Satznummer als zweistellige ganze Dezimalzahl, deren Ziffern ebenfalls Zonen aufweisen. Der Gebrauch dieses Formats ist nur beim Betriebssystem DOS gestattet. Führende Nullen bei der Spurnummer können durch Leerzeichen ersetzt sein.

3) MBBCCHHR [3]

   Bei diesem Format handelt es sich um einen acht Bytes umfassenden nichtzusammenhängenden Dualwert. Unter M ist die Nummer der Speicherbereiche einer Datenmenge auf der SEDZ [4] aufzuführen, 0 steht für den ersten oder einzigen Bereich, 1 für den zweiten usw. BB ist gleich Null für alle von uns betrachteten SEDZ; BB steht für Behälter (Zelle) und kann ungleich Null nur für solche Geräte sein, bei denen mehr als ein Stapel bzw. eine Zelle zur gleichen Zeit montiert sein kann. Für CC ist die Zylindernummer einzusetzen, HH bedeutet die Nummer des Schreib-/Lesekopfes des Zugriffsmechanismus. Schließlich verweist R wie üblich auf die Satznummer.

---

1) Durch jeden Großbuchstaben soll hierbei stets ein Byte symbolisiert werden, sein Inhalt wird dabei durch einen bestimmten Großbuchstaben beschrieben.

2) Unter T bei den ersten beiden Formatarten stets die Spur (vom englischen Wort „track") und unter R immer der Satz (vom englischen Wort „record") verstanden werden.

3) Unter B soll hier die Zelle bzw. Behälter (vom englischen Wort „bin") unter C der Zylinder (vom englischen Wort „cylinder" und unter H der Schreibe/Lesekopf (vom englischen Wort „Head" verstanden werden; R steht nach wie vor für Satz.

4) SEDZ — Speichereinheit mit direkter Zugriffsmöglichkeit

4) NNN

Durch eine drei Bytes lange ganze Dualzahl wird die Blocknummer einer Datenmenge festgelegt, und zwar relativ zum ersten Block, der die Nummer 0 trägt. Nur für Sätze der Satzart F ist eine solche Angabe möglich. Dieses Format kann nur im Betriebssystem OS gebraucht werden, obgleich im Betriebssystem DOS bei Benutzung von PL/1 eine Routine zugänglich ist, die automatisch dieses Format in das Format TTTR (siehe unter 1) umsetzt.

Schlüssel können bei den Formaten 1, 2 und 3 zusätzlich gebraucht werden. Geschieht das, so wird die Satznummer R ignoriert.

Von PL/1 werden die Formate 1 und 4 unterstützt, von COBOL im Betriebssystem DOS die Formate 1, 2 und 3, im Betriebssystem OS hingegen die Formate 1, 3 und 4. Von der AL werden mit Ausnahme des Formates 4 im DOS und des Formates 2 im OS alle anderen Formate bedient. In FORTRAN können keine wahlfrei organisierten Datenmengen angesprochen werden.
Die wahlfreie Organisationsform wird auch als regionale Organisationsform bezeichnet.

10.19. Indexsequentiell (logisch fortlaufend) organisierte Datenmengen

Indexsequentiell (logisch fortlaufend) organisierte Datenmengen schließen einen Index (Inhaltsverzeichnis) ein, der auf einer SEDZ gespeichert sein muß. Um einen Satz zu finden, wird im Inhaltsverzeichnis nachgeschlagen. Dieses verweist auf die Spur, auf der der Satz mit dem gesuchten Schlüssel gefunden werden kann. Die Sätze von indexsequentiell organisierten Datenmengen werden stets mit Schlüsseln aufgezeichnet. Sie können das F- oder das V-Format aufweisen, jedoch niemals das U-Format. In beiden zulässigen Satzarten können sie geblockt oder ungeblockt vorliegen; im allgemeinen sorgt man für ihre Blockung. Sind sie geblockt, so muß das Schlüsselfeld innerhalb des Datenteiles des Satzes wiederholt werden. Für ungeblockte Sätze entfällt diese Notwendigkeit, nichtsdestotrotz wird es auch bei ihnen gewöhnlich wiederholt.

Das Betriebssystem DOS läßt Sätze im V-Format nicht zu. Bei Programmierung in COBOL muß das Schlüsselfeld immer im Datenteil wiederholt werden. Bei Benutzung der FORTRAN-Sprache können indexsequentiell organisierte Datenmengen nicht angesprochen werden.

Um das Konzept der indexsequentiellen Organisationsform zu erklären, wollen wir ungeblockte Sätze fester Länge (F-Format) betrachten. Nehmen wir einmal an, daß 91 Sätze mit einer Länge von 2400 Stellen (Bytes) vorliegen, deren Schlüssel zwei Bytes umfaßt und aus einer positiven Dualzahl besteht. Die soeben umrissene Datenmenge soll die Zylinder 87 und 88 eines Plattenstapels einer IBM 3330 belegen. Teile der Spur 0 und die gesamten Spuren 15, 16, 17 und 18 sollen für Zwecke reserviert bleiben, mit denen wir uns später befassen werden. Wie in der Abb. 10.10. dargestellt, wollen wir die Sätze so verteilen, daß zwei Sätze auf der Spur 0 zu liegen kommen und jeweils fünf auf den übrigen Spuren. Ein Satz, der das Ende der Datenmenge anzeigt (Dateiendesatz DE) soll auf den letzten Datensatz folgen. Bei einer IBM 3330 passen gerade fünf Sätze der angenommenen Länge auf eine Spur.

Die Spuren 15 bis einschließlich 18 sind als Überlaufbereich vorgesehen, d.h. als Bereich, der die Sätze aufnehmen soll, die zur Datenmenge hinzugefügt werden. Außerdem reservieren wir den ersten Teil der Spur 0 für die Speicherung von 31 Sätzen mit einer Länge des Datenteils von 10 Bytes, denen jeweils ein zweistelliges Schlüsselfeld vorausgeht. Der Satzbereich R_0 auf der Spur 0 des Zylinders 87 soll acht Bytes

	Spur	Inhalt				
Zylinder 87	Ø	Indexsätze			2	5
	1	7	11	14	17	19
	2	21	24	29	31	37
	3	39	41	42	43	44
	4	45	46	48	49	5Ø
	5	51	53	59	61	69
	6	71	73	79	83	89
	7	91	93	94	95	96
	8	98	99	1ØØ	1Ø2	1Ø4
	9	1Ø5	1Ø6	1Ø8	1Ø9	111
	1Ø	113	119	123	126	127
	11	129	131	133	137	138
	12	139	14Ø	141	142	143
	13	144	148	15Ø	158	172
	14	176	18Ø	181	183	184

	Spur	Inhalt				
Zylinder 88	Ø	Indexsätze			186	187
	1	188	189	19Ø	192	197
	2	2ØØ	2Ø8	217	24Ø	267
	3	278	3Ø2	3Ø8	317	32Ø
	4	344	438	DE		
	.					
	.					
	.					

Abb. 10.10. Beispiel für eine indexsequentiell (logisch fortlaufend) organisierte Datenmenge

lang sein und den sogenannten Zylinderüberlaufsteuersatz [1] beinhalten. Dieser Satz enthält zunächst einmal die Zeichenfolge CCHHR [2], wobei CC gleich 87, HH gleich 15 und R gleich 1 ist. Damit verweist er auf den ersten Überlaufsatz, der zu diesem Zylinder gehört. In diesen Satz ist außerdem ZZ hineingestellt, ein zweistelliger Zähler, der die Anzahl der Bytes nennt, die auf einer Überlaufspur verfügbar sind; bei der IBM 3330 sind das 13030. Das achte Byte von R_0 wird nicht benutzt.

Der Satzbereich R_1 auf der Spur 0 enthält im Schlüsselfeld den Wert 5, d.h. den höchsten Schlüssel der auf dieser Spur gespeicherten Sätze. Im Datenteil stehen für CCHHR die nachfolgenden Werte: CC gleich 87, HH gleich 0 und R gleich 32. Somit wird durch diesen Indexsatz auf den ersten Datensatz auf dieser Spur verwiesen. Die restlichen fünf Bytes des Datenteils des Indexsatzes identifizieren ihn als solchen.

1) Im angloamerikanischen Sprachraum als „cylinder overflow control record", abgekürzt COCR, bezeichnet

2) Siehe hierzu Abschnitt 10.18.

Sehen wir uns nunmehr die weiteren Indexsätze an. Der Inhalt von R_2 ist identisch mit dem von R_1. R_3 enthält im Schlüsselfeld 19 und im Datenteil wird auf 87,1,1 verwiesen, d.h. der Schlüssel stellt den höchsten Schlüssel der auf der Spur 1 gespeicherten Sätze dar und 87,1,1 sind die Adreßkomponenten (Zylinder, Spur, Satzbereich) des ersten auf dieser Spur gespeicherten Datensatzes. R_4 enthält ein Duplikat von R_3. In gleicher Weise gehören die beiden nächsten Indexsätze (in R_5 und R_6) zur Spur 2, die folgenden beiden zur Spur 3 usw. Schließlich und endlich zeigen die Inhalte von R_{29} und R_{30} zur Spur 14. Alle Bitstellen der beiden Bytes des Schlüsselfeldes des Satzbereiches R_{31} sind auf 1 gesetzt; der Datenteil bleibt unbeachtet. Das aus lauter Einerbits bestehende Schlüsselfeld signalisiert auf diese Weise das Ende des Spurindex (Spureninhaltsverzeichnisses).

Um die Datenmenge zu erstellen, beginnen wir mit dem Schreiben der ersten 31 Sätze auf die Spur 0 des Zylinders 87, wobei wir Scheinwerte verwenden. Danach zeichnen wir die ersten beiden Datensätze in die Spur 0 auf. Nunmehr gehen wir zurück und schreiben die korrekten Indexsätze nach R_1 und R_2. Wir fahren fort mit dem Aufzeichnen der zur Spur 1 gehörenden Datensätze. Anschließend gehen wir wieder zur Spur 0 zurück und schreiben die korrekten Indexsätze nach R_3 und R_4. Dieses Vorgehen setzen wir solange fort, bis der Zylinder 87 gefüllt ist; danach gehen wir zum nächsten Zylinder über.

Um zu einem Satz zuzugreifen, von dem allein der Schlüssel bekannt ist, durchsuchen wir zuerst den Spurindex. Nehmen wir einmal an, daß wir nach dem Satz mit dem Schlüssel 46 fahnden. Ein Durchsuchen des Spurindex mittels des Befehles „Durchsuchen auf Schlüssel gleich oder hoch", dem ein Durchsuchen der Spur nach der Spuradresse vorausgeht, stoppt beim Satzbereich R_9 mit dem Schlüsselfeldinhalt 50. Der in R_9 gespeicherte Indexsatz wird in den Hauptspeicher gelesen. Er verweist auf den Satzbereich 1 der Spur 4. Durch Aufsetzen des Befehls „Suchen Kopf" sorgen wir für die Einstellung des Zugriffsmechanismus auf die Spur 4. Die weiteren Befehle „Durchsuchen auf gleiche Spuradresse" und „Durchsuchen auf Schlüssel gleich oder hoch" werden uns zum gesuchten Satz führen. Wir bekommen die Gleichheitsanzeige beim Satzbereich 2. Durch Lesen des Schlüssels und der Daten bekommen wir den gewünschten Satz in den Hauptspeicher. Würden wir nun feststellen, daß der gelesene Satz einen Schlüssel größer als 46 hat, so könnten wir daraus schließen, daß in der Datenmenge kein Satz mit dem Schlüssel 46 existiert.

Wir wollen nunmehr annehmen, daß wir drei Sätze in die Datenmenge einfügen sollen: Der erste einzufügende Satz besitzt den Schlüssel 36, der zweite den Schlüssel 532 und der dritte den Schlüssel 173.

Um den Satz mit dem Schlüssel 36 einzufügen, durchsuchen wir zunächst das Spureninhaltsverzeichnis auf 36; dieses führt uns zur Spur 2. Auf der Spur 2 finden wir einen Satz mit dem Schlüssel 37, jedoch keinen Satz mit dem Schlüssel 36. Wir lesen deshalb vom Plattenspeicher den Satz mit dem Schlüssel 37 ein und schreiben an seine Stelle Schlüssel und Daten unseres einzufügenden Satzes. Wir analysieren nun den Satzbereich R_0 von Spur 0. Gemäß der in ihm enthaltenen Information schreiben wir den Satz mit dem Schlüssel 37 auf Spur 15, Satzbereich 1. Selbstverständlich müssen wir anschließend den Inhalt des Zylinderüberlaufsteuersatzes, d.h. des Satzbereiches R_0 von Spur 0, berichtigen: Einmal muß jetzt auf die Spur 15, Satzbereich 2 als denjenigen Satzbereich verwiesen werden, der den nächsten Überlaufsatz aufnehmen kann, und zum anderen muß der Zähler korrigiert werden, indem wir die Anzahl der zur Speicherung des Überlaufsatzes benötigten Bytes von seinem Inhalt subtrahieren. Zum Schluß müssen wir noch die Inhalte von R_5 und R_6 des Spurindex auf die neue Spurbelegung abstimmen. Bei R_5 müssen wir das Schlüsselfeld auf 36 ändern, da 36 nunmehr der höchste

auf der Spur 2 vorkommende Satz ist. Bei R_6 belassen wir das Schlüsselfeld auf seinem alten Stand, ändern jedoch den Adreßverweis des Datenteils auf Zylinder 87, Spur 15, Satzbereich 1. Bei zukünftigen Zugriffen zum Satz mit dem Schlüssel 36 werden wir somit auf die Spur 2 und bei zukünftigen Zugriffen zum Satz mit dem Schlüssel 37 auf die Spur 15 geführt. Der auf der Spur 15 stehende Satz wird als Überlaufsatz bezeichnet, da er eigentlich zu Spur 2 gehört, jedoch aus dieser entfernt wurde, weil nicht mehr genügend Platz auf dieser Primärspur vorhanden war.

Wenden wir uns nun dem Problem zu, den Satz mit dem Schlüssel 532 in die Datenmenge einzufügen. Wir finden dabei, daß 438 der höchste vorkommende Schlüssel in der Datenmenge ist. Deshalb ersetzen wir den Endsatz durch den Satz mit dem Schlüssel 532, schreiben einen neuen Endsatz in den unmittelbar folgenden Satzbereich und ändern das Schlüsselfeld in den beiden Indexsätzen auf 532.

Beim Hinzufügen des Satzes mit dem Schlüssel 173 führt uns das Durchsuchen des Spurindex auf einen Stopp beim Satzbereich R_{29}, zu dem der Schlüssel 184 gehört. Da 173 kleiner als der Schlüssel des ersten auf der Spur 14 gespeicherten Satzes ist, wird der Satz mit dem Schlüssel 176 in den Hauptspeicher gelesen und an seine Stelle der neue Satz mit dem Schlüssel 173 geschrieben. Danach lesen wir den nächsten Satz, d.h. den Satz mit dem Schlüssel 180, in den Hauptspeicher und schreiben an seine Stelle den zuvor geholten Satz mit dem Schlüssel 176. Wir fahren auf diese Weise bis zum Spurende fort, die somit nach Abschluß dieses Vorgehens die Sätze mit den Schlüsseln 173, 176, 180, 181 und 183 enthält. Den zuletzt aus der Spur gelesenen übriggebliebenen Satz, d.h. den Satz mit dem Schlüssel 184, schreiben wir als Überlaufsatz in den Satzbereich 2 der Spur 15. Der Spurindex und der Spurenüberlaufsteuersatz werden zum Schluß zwecks Abstimmung auf die neue Speicherung berichtigt.

Es kann der Fall eintreten, daß sich mehr als ein Satz einer Primärspur im Überlaufbereich befindet. Um diese Möglichkeit zu berücksichtigen, stellen wir ein zehn Bytes umfassendes Ergänzungsfeld vor den eigentlichen Datenteil eines jeden Satzes, den wir in den Überlaufbereich schreiben müssen. In dieses Feld setzen wir einen Zeigerwert hinein, d.h. letztlich die Adresse desjenigen Satzbereiches des Überlaufbereiches, in dem der nächste von der gleichen Primärspur stammende Überlaufsatz gespeichert ist. Der Zeigerwert ist gleich Null, wenn von einer Spur nur ein Überlaufsatz stammt. Um bei einem Beispiel zu bleiben, wollen wir in unsere Datenmenge außer den bereits hinzugefügten Sätzen noch einen Satz mit dem Schlüssel 179 hinzufügen. Dieses Hinzufügen führt dazu, daß der Satz mit dem Schlüssel 183 in den Satzbereich R_3 auf der Spur 15 geschrieben wird. Außerdem wird das Ergänzungsfeld des Satzbereiches R_2 der gleichen Spur, in dem sich ja bereits ein Überlaufsatz der Spur 14 befindet, durch einen Zeigerwert gefüllt, der auf Spur 15, Satzbereich R_3 verweist. Der Satzbereich R_1 der Überlaufspur 15, der bekanntlich einen Überlaufsatz von Spur 2 aufgenommen hat, zeigt nach wie vor auf keinen weiteren Satzbereich der Überlaufspuren. Während der zu einer bestimmten Spur gehörende Satz des Spureninhaltsverzeichnisses in seinem Schlüsselfeld stets den höchsten Schlüssel aller Überlaufsätze der Primärspur enthält, verweist der Adreßteil immer auf den Satzbereich des Überlaufbereiches, in dem der aus dieser Spur ausgelagerte Satz mit dem niedrigsten Schlüssel steht. Wenn wir somit nach dem Satz mit dem Schlüssel 183 suchen, werden wir zunächst auf den Satzbereich R_2 der Spur 15 geführt. Wir lesen den dort stehenden Satz und verwenden den Zeigerwert zur weiteren Suche, nachdem wir festgestellt haben, daß der gelesene Satz nicht der gesuchte ist. Dieser leitet uns schließlich zu R_3. Den von diesem Bereich gelesenen Satz können wir dann leicht als den gesuchten identifizieren.

Da sich eine Datenmenge über mehrere Zylinder, ja sogar über mehrere Plattenstapel ausdehnen kann, müssen wir uns auch ein Zylinderinhaltsverzeichnis (einen

Zylinderindex) aufbauen, den wir sinnvoll auf einem anderen Teil des Plattenspeichers speichern werden. Der Zylinderindex sieht genau so wie ein Spurindex aus, nur daß sich jeweils die beiden zusammengehörenden Sätze auf den Satz mit dem höchsten Schlüssel innerhalb eines Zylinders beziehen. Ein Zylinderindex kann soviele Spuren belegen, wie für ihn als erforderlich angesehen werden. Bei unserem Beispiel benötigen wir den für Zylinderindex 5 Sätze: Der Inhalt von R_1 und R_2 beschreibt den Inhalt des Zylinders 87, der von R_3 und R_4 den Inhalt des Zylinders 88, während R_5 einem Schlüssel gehört, der auf allen Bitstellen eine 1 enthält. Durch R_5 wird somit in üblicher Weise das Ende des Zylinderinhaltsverzeichnisses angezeigt.

Werden mehr als vier Spuren für den Zylinderindex benötigt, so ist es mitunter ratsam, außerdem ein Hauptinhaltsverzeichnis (einen Hauptindex) anzulegen. Praktisch bedeutet dies nichts anderes als ein Inhaltsverzeichnis, das über den Inhalt des Zylinderinhaltsverzeichnisses Auskunft gibt. Ebenso ist das Zylinderinhaltsverzeichnis nichts anderes als ein Verzeichnis, das über den Inhalt des Spureninhaltsverzeichnisses Auskunft gibt.

Wie wir gesehen haben, belegen die Haupt- und Zylinderinhaltsverzeichnisse ihre eigenen Plattenspeicherbereiche. Die Informationen darüber sind in einem Kennsatz des Formates 2 enthalten, der bei allen logisch fortlaufend organisierten Datenmengen an den Kennsatz des Formates 1 gekettet ist. Beide Kennsätze werden zu der Zeit erstellt, zu der die Datenmenge aufgebaut wird.

Da auch die Möglichkeit nicht auszuschalten ist, daß die Überlaufbereiche eines oder mehrerer Zylinder vollständig ausgeschöpft sind, kann zusätzlich ein unabhängiger Zylinderüberlaufbereich eingerichtet werden. Dieser Bereich wird zur Speicherung derjenigen Sätze benutzt, die nicht mehr in die Spurenüberlaufbereiche hineinpassen.

Um einen Satz aus der Datenmenge zu entfernen, können wir ihn nicht einfach aus der Datenmenge nehmen, da diese sonst eine Lücke aufweisen würde. Stattdessen markieren wir den zu entfernenden Satz, indem wir alle Bitstellen des linksbündigen Bytes des Datenteils des Satzes auf 1 setzen. Wird nun ein logisch fortlaufender Datenbestand sequentiell verarbeitet, so stellen die Routinen der Zugriffsmethode einen solchen Satz nicht zur Verarbeitung bereit. Markierte Sätze werden allerdings bei Vorliegen eines gewissen Umstandes durch andere Sätze ersetzt. Betrachten wir den Fall, daß eine Datenmenge durch Zugänge ergänzt werden soll. Die Routinen der Zugriffsmethode können dabei in eine Phase eintreten, die es erforderlich macht, Sätze von einem Satzbereich in einen nachfolgenden Satzbereich weiterzuschieben. Werden bei dieser Verschiebung markierte Sätze entdeckt, so werden diese durch den Zugang bzw. durch den zu verschiebenden Satz ersetzt. Auf diese Weise ist die Notwendigkeit eliminiert, einen weiteren Satz in den Überlaufbereich zu übertragen.

Meist wird nach einer Weile eine aktive, durch viele Neuzugänge und Abgänge beeinflußte Datenmenge gedrängt volle Überlaufbereiche aufweisen. Konsequenterweise wird man deshalb eine solche Datenmenge nach einer gewissen Periode einer Reorganisation unterziehen. Diese besteht letztlich darin, daß man die Sätze fortlaufend liest und woanders wieder aufzeichnet. Da das Lesen die Sätze in sequentieller Folge zur Verfügung stellt und das Schreiben einer Neuerstellung der Datenmenge gleichkommt, besitzt die regenerierte Datei ganz andere Inhaltsverzeichnisse als die ursprüngliche. Da sie praktisch eine völlig neue Datenmenge darstellt, besitzt sie folglich auch keine Überlaufsätze mehr.

Gerade, weil es sich mit einer logisch fortlaufend organisierten Datenmenge so bequem arbeiten läßt, sollte darauf hingewiesen werden, daß bei einer solchen in einem gewis-

sen Umfang auch Plattenspeicherplatz vergeudet wird, da sowohl zusätzlicher Platz für die Inhaltsverzeichnisse als auch für die Überlaufbereiche vorgesehen werden muß. Außerdem erfordert jeder Zugriff zu einem Satz wenigstens zwei Suchvorgänge, wodurch der Verarbeitungsprozeß verlangsamt wird. Als außerordentlich gut ist jedoch die Tatsache hervorzuheben, daß die Routinen, die für die laufende Bestandsführung (Zugänge, Abgänge, Umorganisation) benötigt werden, höchst einfach sind.

Logisch fortlaufend organisierte Datenmengen können auch mit geblockten Sätzen gebildet werden. Hierbei ist es jedoch notwendig, daß der Schlüssel des Satzes unbedingt im Datenteil des Satzes eingebettet ist. Er wird dann Satzschlüssel genannt. Das Blocken der Sätze führt zur Einsparung von Speicherplatz auf den SEDZ. Die Spurindizes verweisen nach wie vor auf den höchsten Schlüssel einer Spur. Der für jeden Block aufgezeichnete Schlüssel entspricht dem höchsten Satzschlüssel im Block. Wenn also ein Block die Sätze mit den Schlüsseln 113, 119 und 123 enthält, so weist das vor diesem Block aufgezeichnete Schlüsselfeld den Inhalt 123 auf. Bei der Verarbeitung geblockter Sätze lesen die EA-Routinen jeweils den gesamten Block in den Hauptspeicher ein. Anschließend wird ein Vergleich zwischen dem Suchschlüssel und den Satzschlüsseln durchgeführt. Bei Übereinstimmung wird der betreffende Schlüssel und der zugehörige Satz dem Benutzerprogramm zur Verfügung gestellt. Werden Sätze zu einer aus geblockten Sätzen bestehenden Datenmenge hinzugefügt, so gleicht die Verfahrensweise der bisher für ungeblockte Sätze besprochenen. Die in Überlaufbereiche hineingeschriebenen Sätze bleiben jedoch stets ungeblockt.

Um zu einem Teil einer Datenmenge zugreifen zu können, wird mitunter ein sogenannter Gattungsschlüssel benutzt. Dieser führt zu einem Suchschlüssel, der nur aus den ersten Bytes des Satzschlüssels besteht. Die Anzahl der in den Vergleich einzubeziehenden Bytes ist im Benutzerprogramm festzulegen. Der erste bei diesem Vorgehen zur Verfügung gestellte Satz ist derjenige, dessen Satzschlüssel gleich oder größer als der Suchschlüssel ist, wobei noch einmal betont werden sollte, daß nur die im Benutzerprogramm festgelegte Anzahl von Bytes in den Vergleich mit dem Suchschlüssel eingeht. Sicher wird der Benutzer in einigen Fällen auf Gleichheit ausgehen müssen.

Das Hauptargument gegen die Benutzung von indexsequentiell organisierten Datenmengen mit einer IBM /370 besteht darin, daß die erforderlichen Kanalprogramme zu einer ziemlich dürftigen Durchflußleistung führen, wenn vom virtuellen Speicher Gebrauch gemacht wird. Ein Weg, dieses Problem zu umgehen, führt dazu, daß man das Programm vollständig im Realspeicher laufen läßt. Ein anderer Weg wird in den folgenden Abschnitten angedeutet.

10.20. Virtuell sequentiell organisierte Datenmengen

Zu der Zeit, als das System IBM /360 im Jahre 1964 eingeführt wurde, wurden auf den damals im Gebrauch befindlichen Computern meist Sätze im F- oder U-Format auf Magnetbändern verarbeitet. Bei der überwiegenden Anzahl von Installationen waren sogar einige Datenmengen zusammengestellt, die geblockte Sätze im F-Format beinhalteten. Auf Magnetplattenspeichern lagen ebenfalls konsekutiv organisierte Datenmengen, aber auch wahlfrei organisierte.

Die Freigabe der Betriebssysteme für die IBM /360 im Jahre 1965 führte zur Fixierung der damals benutzten Form der Sätze im V- und im S-Format. Wahlfrei organisierte Datenmengen erhielten zur damaligen Zeit ebenfalls ihr gegenwärtiges Aussehen. Da damals auch die Verarbeitung von Datenmengen über Inhaltsverzeichnisse erforderlich war, wurden für Computer früher Entwicklungsstadien Programme geschaffen, die auf der Basis wahl-

frei organisierter Datenmengen arbeiteten. Die Einführung der indexsequentiellen Organisationsform bedeutete deshalb einen echten Schritt vorwärts: Viele Programme konnten wesentlich einfacher gestaltet werden. Allerdings warf die indexsequentielle Organisationsform eine Reihe neuer Probleme auf: Die dieser Organisationsform unterliegenden Datenmengen mußten periodisch reorganisiert werden, um Speicherplatz wiederzugewinnen und um die miserable Durchflußleistung bei sequentiellen Zugriffen zu verbessern, besonders nach dem Hinzufügen vieler neuer Sätze.

Das Führen des gesamten Zylinderinhaltsverzeichnisses oder eines Teils desselben im Arbeitsspeicher, d.h. im Hauptspeicher, kam auf, wodurch ein praktikabler Weg aufgezeigt war, der geeignet war, die Leistung zu verbessern. Bei Computern mit virtuellen Speichern erbrachten die für den Zugriff zu indexsequentiell organisierten Datenmengen entwickelten Kanalprogramme eine relativ schlechtere Leistung als die für andere Organisationsformen geschaffenen. Deshalb wurde eine neue Zugriffsmethode entwickelt, die virtuell sequentielle Zugriffsmethode, kurz auch VSAM [1] genannt. Sie ist dazu bestimmt, für ein standardisiertes Vorgehen bei der Behandlung von nahezu allen auf SEDZ gespeicherten nichttemporären Datenmengen zu sorgen. Eine solche Zugriffsmethode setzt natürlich auch eine bestimmte Organisation der Datenmenge voraus. Die physischen Eigenschaften der Datenmenge, wie maximale Satzlänge, Platz des Schlüssels innerhalb der Sätze, Größe der kleinstmöglichen EA-Gebiete, die beim Lesen von Sätzen aus der Datenmenge verwendet werden können, und andere Angaben, werden bei der Erstellung der Datenmenge im VSAM-Katalog gespeichert. Dadurch vereinfachen sich die Programme, die zu virtuell sequentiell organisierten Datenmengen zugreifen, da die zur Verarbeitung der Datenmengen benötigten Informationen meist aus dem Katalog entnommen werden können. Es ist nicht nötig, sie im Anwendungsprogramm mitzuführen.

Auf zwei verschiedene Weisen können Datenmengen virtuell sequentiell organisiert werden. Eine Datenmenge in Zugangsfolge, abgekürzt ESDS [2], ist im Grunde genommen eine verbesserte Version der herkömmlichen konsekutiven Organisationsform. Eine Datenmenge in Schlüsselfolge, abgekürzt KSDS [3], verhält sich wie eine indexsequentiell organisierte Datenmenge. Eine Datenmenge in Zugangsfolge besitzt die Eigenschaft, daß Zeiger zu individuellen Sätzen wirkungsvoll benutzt werden können, um wahlfrei Sätze wiederzubekommen. Somit gestattet eine Datenmenge in Zugangsfolge nicht nur die Anwendung der klassischen sequentiellen Zugriffsmethode, sondern auch die Anwendung direkter Zugriffsmethoden. Dadurch, daß alle Sätze virtuell sequentiell organisierter Datenmengen geblockt vorliegen, wird ein wesentlicher Gewinn gegenüber den wahlfrei organisierten Datenmengen erzielt, die nur ungeblockte Sätze aufweisen können.

Bei einer Datenmenge in Schlüsselfolge werden die Datensätze genau so wie bei einer Datenmenge in Zugangsfolge gespeichert. Sie kann infolgedessen ebenso verarbeitet werden wie eine Datenmenge in Zugangsfolge. Die Schlüssel, durch die auf die Datensätze verwiesen wird, werden in einer anderen Teildatenmenge in Schlüsselfolge gespeichert. Die Teildatenmenge mit den Schlüsseln enthält die Äquivalente zu den Spurindexsätzen, den Zylinderindizes und dem Hauptindex von indexsequentiell organisierten Datenmengen. Obgleich die periodische Reorganisation einer Datenmenge in Schlüsselfolge nach wie vor notwendig ist, um verlorenen Speicherplatz wiederzugewinnen, bedeutet das Vorhandensein vieler Zugänge in der Datenmenge fast keinen Leistungsverlust mehr.

---

1) VSAM ist die Abkürzung von „virtual sequential access method".

2) ESDS ist die Abkürzung von „entry sequenced data set"

3) KSDS ist die Abkürzung von „key sequenced data set"

Eine spezielle VSAM-ISAM-Schnittstellenroutine [1] ermöglicht es, in Programmen, die für die Handhabung von indexsequentiell organisierten Datenmengen geschrieben wurden, die Verarbeitung von virtuell sequentiell organisierten Datenmengen vorzunehmen. Dabei werden die Sätze dem Benutzer so zur Verfügung gestellt, als ob es sich um Sätze aus einer indexsequentiell organisierten Datenmenge handelt. Bei indexsequentiell organisierten Datenmengen, bei denen die Schlüssel nicht in die Sätze eingebettet sind, stellt die Schnittstellenroutine den Schlüssel an den Anfang des für die Eingabe vorgesehenen Arbeitsgebietes, unmittelbar vor den Satz also. Gewisse Fehleranzeigen, die bei einer indexsequentiell organisierten Datenmenge auftreten können, können bei Benutzung der Schnittstellenroutine nicht erscheinen.

Virtuell sequentiell organisierte Datenmengen werden in den Programmiersprachen PL/1, COBOL und AL [2] unterstützt. In der FORTRAN-Sprache sind sie z. Z. noch nicht zugänglich.

Wie bei einer indexsequentiell organisierten Datenmenge kann auch hier ein Gattungsschlüssel benutzt werden, d. h. der Schlüsselvergleich erstreckt sich nur auf den ersten Teil des aufgezeichneten Schlüssels. Ein Gattungsschlüssel kann beispielsweise dazu verwendet werden, den Startpunkt festzulegen, von dem auf der restliche Teil der Datenmenge sequentiell verarbeitet werden soll.

Wenn eine virtuell sequentiell organisierte Datenmenge erzeugt wird, kann bei dieser Gelegenheit ihr Schutz durch ein Paßwort vereinbart werden. Verschiedene Paßwörter können erforderlich sein, wenn einmal die Sätze der Datenmenge bloß gelesen werden sollen, zum anderen, wenn existierende Sätze geändert werden sollen. Über Paßwörter wollen wir jetzt nicht weiter diskutieren.

10.21. Einige Aspekte zum Aufbau virtuell sequentiell organisierter Datenmengen

Virtuell sequentiell organisierte (VSO) Datenmengen werden nicht in Blöcke geschrieben. Sie werden stattdessen in sogenannte Kontrollintervalle eingefügt. Ein Kontrollintervall besitzt die in Abb. 10.11. dargestellte Form.

1. Satz	2. Satz	3. Satz	...	n. Satz	Freispeicher	Kontrollinformationsbereich

Abb. 10.11. Kontrollintervall von VSO-Datenmengen

Die Länge der Kontrollintervalle kann zwischen 512 und 32768 Bytes in Vielfachen von 512 festgelegt werden; sind sie größer als 8192 Bytes, so muß ihre Länge ein Vielfaches von 2048 sein. Bei einer Datenmenge in Schlüsselfolge dürfen die Schlüsselsätze nur in Kontrollintervallen untergebracht werden, deren Umfang 4096 Bytes nicht überschreitet.

Die Datensätze beginnen am Anfang des Kontrollintervalles. Dabei fängt jeder Satz unmittelbar hinter dem Ende des vorhergehenden Satzes an, d. h. sein erstes Zeichen belegt das auf das Ende folgende erste Byte. Die am weitesten rechts angeordneten vier Bytes des Kontrollinformationsbereiches enthalten ein Kontrollintervalldefinitionsfeld, abgekürzt CIDF[3]. Das erste Halbwort dieses Feldes

1) ISAM ist die Abkürzung von „indexsequential access method", d. h. der Zugriffsmethode für indexsequentiell organisierte Datenmengen

2) AL ist die Abkürzung von „Assembler Language"

3) CIDF ist die Abkürzung von „control interval definition field"

enthält als ganze Dualzahl die Anzahl der von den Datensätzen okkupierten Bytes im Kontrollintervall. Enthält das Kontrollintervall keine Daten, so ist diesem Halbwort der Wert 0 zugewiesen. Das letzte Halbwort hingegen enthält die Anzahl der unbelegten Bytes, d.h. den Umfang des Freispeichers. Diese ist gleich der Länge des Kontrollintervalls vermindert um die Summe aus der Anzahl der Datenbytes und der Steuerinformationen. Wenn kein freier Bereich vorhanden ist, ist dieser Wert selbstverständlich gleich Null.

Unter dem Freispeicher wird derjenige Teil des Kontrollintervalls verstanden, der nicht benutzt ist. Bei einer Datenmenge in Zugangsfolge entsteht ein Freispeicher, wenn der nächste zu schreibende Satz umfangreicher ist als der im gegenwärtig benutzten Kontrollintervall verbliebene Platz. Bei einer Datenmenge in Schlüsselfolge kann ein bestimmter Prozentsatz des Kontrollintervalls als Freispeicher reserviert bleiben, wenn die Datenmenge erzeugt wird. Auf diese Weise wird das spätere Hinzufügen neuer Sätze erheblich vereinfacht.

Neben dem Kontrollintervalldefinitionsfeld enthalten die Steuerinformationen noch drei Bytes lange Satzdefinitionsfelder, abgekürzt RDF [1]. Das erste Byte jedes RDF stellt ein Kennzeichen mit folgender Bedeutung dar:

X'ØØ'	Die nächsten beiden Bytes enthalten die Länge eines einzelnen Satzes
X'4Ø'	Die nächsten beiden Bytes enthalten die Länge von mehreren gleichlangen Sätzen
X'Ø8'	Die nächsten beiden Bytes enthalten die Anzahl der Sätze gleicher Länge

Die Datensätze werden im Kontrollintervall von links nach rechts gespeichert, während die zugehörigen Satzdefinitionsfelder von rechts nach links angeordnet werden. Besitzt mehr als ein Datensatz die gleiche Länge, so braucht nicht für jeden Satz ein RDF generiert zu werden. Es genügt, ein RDF-Paar für alle aufeinanderfolgenden Sätze der gleichen Länge einzurichten. Das rechte Feld dieser beiden besitzt als Kennzeichen X'4Ø', das linke das Kennzeichen X'Ø8'.

Um die Belegung eines Kontrollintervalls zu illustrieren, wollen wir ein Beispiel betrachten. Der Umfang der Kontrollintervalle soll 512 Bytes betragen. Die zu speichernden Sätze sollen in Zugangsfolge untergebracht werden. Es sind neun Sätze zu speichern, die die folgenden Längen besitzen:

Satz	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
Länge in Bytes	100	100	30	120	120	120	120	120	120

Die ersten fünf Sätze passen in das erste Kontrollintervall. Der 6. Satz ist zu lang, um noch vom ersten Kontrollintervall aufgenommen werden zu können. Deshalb beginnt mit ihm das Füllen des zweiten Kontrollintervalls. Die beiden Kontrollintervalle sind nach der Einspeicherung der neun Sätze so belegt, wie es die Abb. 10.12. zeigt.

Für jeden Satz existiert ein Zeiger. Der Zeiger ist einfach die relative Byteadresse des ersten Bytes des betreffenden Satzes, relativ zum Beginn der Datenmenge. Der Zeiger wird als ganze, ein Wort belegende Dualzahl ausgedrückt. Für S_1 ist der Zeiger gleich 00000000, für S_2 gleich 00000064, für S_6 gleich 00000200 usw. Durch Angabe eines Zeigers kann zu jedem Satz direkt zugegriffen werden.

Der Datenteil einer Datenmenge in Schlüsselfolge ist identisch mit einer Datenmenge in Zugangsfolge. Die einzigen Beschränkungen sind folgende: a) Jeder Satz muß einen

1) RDF ist die Abkürzung von „record definition field"

Kontrollinvervall 1							Kontrollinformationsbereich				
Inhalt	S_1	S_2	S_3	S_4	S_5	Freispeicher	RDF_4	RDF_3	RDF_2	RDF_1	CIDF
Länge in Bytes	100	100	100	30	120	46	3	3	3	3	4

Kontrollintervall 2						Kontrollinformationsbereich		
Inhalt	S_6	S_7	S_8	S_9	Freispeicher	RDF_2	RDF_1	CIDF
Länge in Bytes	120	120	120	120	22	3	3	4

Inhalt der Kontrollinformationsbereiche der Kontrollintervalle

Kontrollintervall 1	
Feld	Sedezimaler Inhalt
RDF_4	ØØ ØØ78
RDF_3	ØØ ØØ1E
RDF_2	Ø8 ØØØ3
RDF_1	4Ø ØØ64
CIDF	Ø1C2 ØØ2E

Kontrollintervall 2	
Feld	Sedezimaler Inhalt
RDF_2	Ø8 ØØØ4
RDF_1	4Ø ØØ78
CIDF	Ø1EØ ØØ16

Abb. 10.12. Beispiel für die Belegung zweier Kontrollintervalle durch eine aus neuen Sätzen bestehende Datenmenge in Zugangsfolge

Schlüssel aufweisen, b) alle Schlüsselfelder müssen innerhalb des Satzes an der gleichen relativen Stelle beginnen, c) die Schlüssel müssen bei allen Sätzen die gleiche Länge aufweisen. Die Schlüsseldatenmenge wird zur gleichen Zeit aufgebaut, zu der die Datensätze geschrieben werden. Um zu Sätzen zuzugreifen, können an Stelle der Schlüssel auch Zeiger benutzt werden. Dabei muß man jedoch darauf achten, daß durch Entfernungen und Hinzufügungen der physische Platz des gesuchten Satzes und damit auch sein Zeiger geändert sein kann.

Unter einem Haufen[1)] versteht man entweder eine Datenmenge in Zugangsfolge oder die beiden Teildatenmengen, die zusammen eine Datenmenge in Schlüsselfolge bilden. Es ist möglich, die beiden Teildatenmengen einer Datenmenge in Schlüsselfolge getrennt zu verarbeiten, doch wollen wir darauf verzichten, weil eine solche Verarbeitung meist sinnlos erscheint.

Der für einen Haufen bereitgestellte Platz auf SEDZ ist bei Datenmengen in Schlüsselfolge auf die beiden Teildatenmengen aufzuteilen. Für jede einzelne Datenmenge wiederum ist der Platz in sogenannte Kontrollgebiete aufzuteilen. Der Umfang der Kontrollgebiete ist bei der Definition des Haufens festzulegen. Er darf einen Zylinder nicht überschreiten und besteht immer aus einer ganzzahligen Anzahl von Spuren. Ein Kontrollgebiet muß mindestens drei Kontrollintervalle enthalten, gewöhnlich werden es mehr sein. Die Leistungsfähigkeit von VSAM wird beinahe immer dann verbessert,

1) Im angloamerikanischen Sprachraum als „cluster" bezeichnet

wenn als Kontrollgebiet ein voller Zylinder vorgesehen ist. Es besteht selten ein Grund, ein kleineres Gebiet zu wählen. Bei einer Datenmenge in Schlüsselfolge wird für die Teildatenmenge der Schlüssel nur ein Kontrollgebiet benutzt.

Wenn ein Haufen definiert wird, so wird ihm Speicherplatz zugeordnet. Sollte es sich erweisen, daß dieser nicht ausreicht, so können nach Bedarf zusätzliche Kontrollgebiete zugeordnet werden. Bei Datenmengen in Zugangsfolge geschieht dies stets am Ende der Datenmenge. Bei Datenmengen in Schlüsselfolge kann in jedem Kontrollgebiet ein gewisser Prozentsatz der Kontrollintervalle ungenutzt gelassen werden, um damit Raum für Hinzufügungen zu schaffen. Bewirken Hinzufügungen von Sätzen, daß ein Kontrollintervall überläuft, so wird ein neues Kontrollintervall benutzt, wobei die bisher eingespeicherten Sätze auf beide Kontrollintervalle verteilt werden. Läuft ein Kontrollgebiet über, so wird ein neues eingerichtet, wobei die Hälfte der Sätze in das neue Kontrollgebiet übergeht.

Es ist möglich, Platz auf der ersten Spur eines Kontrollgebietes für diejenigen Schlüsselsätze zu reservieren, die bei der indexsequentiellen Organisationsform den Spurindex bilden. In manchen Fällen kann dadurch die Leistungsfähigkeit von VSAM verbessert werden. Die Entscheidung hierüber muß vor der Generierung der Datenmenge gefällt werden. Es ist ebenfalls möglich, die gesamte erste Spur jedes Kontrollgebietes für Schlüsselsätze bereitzustellen. In diesem Fall werden die Schlüsselsätze so oft auf der Spur zyklisch wiederholt, wie dafür der Platz ausreicht. Damit kann sogar eine bessere Leistung erzielt werden, wenn nach einem Schlüsselsatz gesucht wird. Werden Schlüsselsätze in das Datengebiet aufgezeichnet, so können sie, aber brauchen nicht in der separaten Teildatenmenge für die Schlüssel aufgezeichnet zu werden. Schlüsselsätze, die dem Typ der Zylinder- und Hauptindexsätze der indexsequentiellen Organisationsform entsprechen, dürfen jedoch niemals ins Datengebiet geschrieben werden.

10.22. VSAM-Dienstleistungen

Virtuell sequentiell organisierte Datenmengen werden im Gegensatz zu den Datenmengen anderer Organisationsformen nicht durch normale Benutzerprogramme angelegt oder zerstört. Stattdessen wird ein von der IBM unterstütztes und gewartetes Programm mit Namen IDCAMS benutzt. Eine vollständige Besprechung von IDCAMS liegt jenseits unserer Bedrüfnisse. Wir wollen uns deshalb mit den Vorgaben für dieses Programm begnügen, die es uns ermöglichen,

a) Speicherraum zu definieren und zu zerstören
b) Haufen anzulegen und zu beseitigen
c) Haufen zu drucken.

Unter einem Speicherraum wollen wir einen Bereich auf einer SEDZ verstehen, der zur Aufnahme von virtuell sequentiell organisierten Datenmengen vorgesehen ist; um Raum zu reservieren, haben wir zu codieren:

```
DEFINE SPACE(VOLUMES(sn [,sn]...)
             FILE(dname)
             CYLINDERS(prim[sek]))
```

Unter dname ist der Dateiname aufzuführen, d.h. im Betriebssystem OS der ddname (siehe Abschnitt 10.15.) und im Betriebssystem DOS der dateiname (siehe Abschnitt 10.9.), der zu den Datenträgern mit den Kennungen (Reihennummern) sn··· gehört. Der zu reservierende Gesamtraum wird hier auf Zylinder bezogen und ist als ganze Zahl unter prim aufzuführen. Sollte es sich nachträglich erweisen, daß dieser Raum

nicht ausreicht, so werden zusätzlich Sekundärzylinder bereitgestellt; deren Zahl ist unter sek aufzuführen. Eine typische Steuerstatementfolge zur Lösung dieser Aufgabe ist für die beiden Betriebssysteme DOS und OS in der Abb. 10.13. aufgeführt.

Betriebssystem DOS

```
// JOB     SPACE
// ASSGN   SYSØØ7,333Ø,VOL=222222
// DLBL    EXSP,,,VSAM
// EXTENT  SYSØØ7,222222,1,Ø,19Ø,19ØØ
// EXEC    IDCAMS,SIZE=1ØØK
   DEFINE  SPACE(VOLUMES(222222)-
           FILE(EXSP)-
           CYLINDERS(1ØØ 2))
```

Betriebssystem OS

```
//SPACE    JOB ···
//STEP     EXEC PGM=IDCAMS
//EXSP     DD DISP=OLD,UNIT=333Ø,VOL=SER=222222
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
  DEFINE   SPACE(VOLUMES(222222)-
           FILE(EXSP)CYLINDERS(1ØØ 2))
/*
```

Anmerkung: Als Parameter des Dispositionsoperanden im DD-Statement ist stets OLD zu benutzen.

Abb. 10.13. Betriebssysteme DOS und OS: Job zur Reservierung von Speicherraum v für eine VSO-Datenmenge

Die Eingabe für das Programm IDCAMS kann die Spalten 2 bis einschließlich 72 belegen Um einen Befehl für dieses Programm über eine Zeile (Lochkarte) hinweg fortsetzen zu können, muß auf der fortzusetzenden Zeile als letztes Zeichen der Bindestrich (Minuszeichen) erscheinen. Um einen Parameter über eine Zeile hinweg weiterzuführen, ist stattdessen das Pluszeichen zu verwenden; am Beispiel

```
CYLINDERS(1ØØ +
          2
```

sei das demonstriert. Unterparameter können durch ein Komma oder durch ein oder mehrere Leerzeichen voneinander getrennt werden. Somit sind die Schreibformen

CYLINDERS(1ØØ,2) und CYLINDERS(1ØØ 2)

einander gleichwertig. Soll der Speicherraum auf Spuren bezogen werden, so ist stattdessen

TRACKS(prim sek)

zu codieren. Wird sek weggelassen, so erfolgt keine sekundäre Zuordnung.

Beim Betriebssystem DOS kann nicht ganz so freizügig vorgegangen werden, wie es soeben beschrieben wurde. Obgleich die Angaben CYLINDERS oder TRACKS erforderlich sind, wird die Angabe für den zuzuordnenden Speicherraum aus dem EXTENT-Statement der Jobsteuersprache genommen.

Um einen Haufen zu definieren, muß das in Abb. 10.14. dargestellte Statement codiert werden.

Der unter NAME aufgeführte Name geht in den VSAM-Katalog ein. SUBALLOCATION (als Standardangabe ist sie unterstrichen) bedeutet, daß der Speicherraum für den Haufen von dem Speicherraum zu nehmen ist, der zuvor durch ein Statement DEFINE SPACE reserviert wurde; UNIQUE hingegen sagt aus, daß der Speicherraum

```
DEFINE  CLUSTER (NAME(name)
                [VOLUMES(sn [sn] ···)]
                [CYLINDERS (prim [sek])  ]
                [TRACKS(prim [sek])      ]
                [SUBALLOCATION           ]
                [UNIQUE                  ]
                [FILE(dname)]
                [RECORDSIZE(durch max)]
                [CONTROLINTERVALSIZE(umfang)]
                [BUFFERSPACE(umfang)]
                [TO(jjttt)               ]
                [FOR(tttt)               ]
                [INDEXED                 ]
                [NONINDEXED              ]
                [KEYS(länge position)]
                [FREESPACE(ki [kg])]
        [DATA (angd)]
        [INDEX (angi)]
```

Anmerkung: Sind Schlüsselwörter (in Großbuchstaben geschriebene Wörter) unterstrichen, so gelten sie als Standard, d.h. bei Weglassung der alternativen Schlüsselwörter werden sie angezogen.

Abb. 10.14. Definition eines Haufens

jetzt angelegt werden soll. Die Angabe FILE wird bei Angabe von UNIQUE benötigt, für SUBALLOCATION ist ihre Notiz freigestellt. Bei der Angabe RECORDSIZE wird unter durch die durchschnittliche Satzlänge und unter max die maximale Satzlänge angezeigt. Fehlt die Angabe für den Umfang des Kontrollintervalls (CONTROLINTERVALSIZE), so wird standardmäßig für die Datensätze 2048 und für die Schlüsselsätze 512 benutzt; erscheint diese Angabe, so muß umfang ein Vielfaches von 512 sein, oder falls größer als 8192, ein Vielfaches von 2048. Durch BUFFERSPACE wird der minimale Speicherplatz angegeben, den ein Programm, das diesen Haufen anspricht, für Puffer bereitstellen kann; der hier aufgeführte Wert darf nicht kleiner als der Standardwert sein und dieser ist gleich der Summe aus dem Zweifachen des Umfanges des Kontrollintervalls für die Daten und (bei Datenmengen in Schlüsselfolge) dem Umfang des Kontrollintervalles für die Schlüssel. Das Verfallsdatum für einen Haufen kann entweder in der Form jjttt (j Jahr, t laufender Tag des Jahres) oder in der Form tttt (t Tage vom Generierungsdatum an gerechnet) aufgeführt werden; Standard sind 0 Tage. INDEXED bedeutet eine Datenmenge in Schlüsselfolge, NONINDEXED eine in Zugangsfolge. KEYS und FREESPACE werden nur für eine Datenmenge in Schlüsselfolge benötigt. Durch länge und position wird bei KEYS die Länge und die Lage des Schlüsselfeldes innerhalb des Satzes angegeben. FREESPACE besagt, daß ki Prozent des in jedem Kontrollintervall zur Verfügung stehenden Platzes für zukünftige Hinzufügungen reserviert bleiben soll und daß kg Prozent der Kontrollintervalle jedes Kontrollgebietes für Hinzufügungen gedacht sind. Die Standardwerte sind Null.

Unter angd sind die Angaben aufzuführen, die nur für die Datenkomponente innerhalb eines Haufens gelten. Sie sind an Stelle von angd in Klammern zu notieren. Hierzu zählen die bereits besprochenen Parmeter VOLUMES, CYLINDERS, TRACKS, SUBALLOCATION, UNIQUE, FILE, RECORDSIZE, CONTROLINTERVALSIZE, BUFFERSPACE, KEYS und FREESPACE. Angaben, die nur für die Schlüsselkomponente innerhalb eines Haufens gelten sollen, sind in Klammern dem Schlüsselwort INDEX unter angi aufzuführen. Hierzu zählen solche Parameter wie VOLUMES, CYLINDERS, TRACKS, SUBALLOCATION, UNIQUE, FILE und CONTROLINTERVALSIZE.

Die Parameter FILE, VOLUMES, TRACKS oder CYLINDERS und RECORDSIZE sowie KEYS für eine Datenmenge in Schlüsselfolge müssen entweder für den Haufen selbst oder für die Datenkomponente spezifiziert werden; FILE, VOLUMES und TRACKS oder CYLINDERS müssen auch für die Schlüsselkomponente einer Datenmenge in Schlüsselfolge erscheinen, wenn sie für die Datenkomponente getrennt definiert sind.

Im Betriebssystem DOS wird für eine Datenmenge mit der Angabe UNIQUE der Speicherraum angelegt, der aus dem EXTENT-Statement der Jobsteuersprache ersichtlich ist. Separate FILE- und TRACKS/CYLINDERS-Angaben müssen für jede Komponente des Haufens gemacht werden. Außerdem müssen die Bereiche auf den SEDZ an Zylindergrenzen beginnen und enden. Man sollte sich aus diesen Einschränkungen die Tatsache vergegenwärtigen, daß damit die Schaffung zweier Paare von DLBL- und EXTENT-Statements verbunden ist.

Einige typische Definitionen von Haufen sind als Beispiele in der Abb. 10.15. aufgeführt.

```
DEFINE  CLUSTER (NAME(ESDS.A)-
                 VOLUMES(222222) CYLINDERS(3) RECORDSIZE(48,48)-
                 CONTROLINTERVALSIZE(4Ø96) NONINDEXED)

DEFINE  CLUSTER (NAME(KSDS.B)-
                 FOR(8) KEYS(3,Ø) FREESPACE(2Ø,1Ø)-
                 RECORDSIZE(176,4Ø64) UNIQUE-
                 BUFFERSPACE(128ØØ))
        DATA (TRACKS(19) CONTROLINTERVALSIZE(6144)-
              FILE(BSP2) VOLUMES(222222))
        INDEX (CYLINDERS(1) CONTROLINTERVALSIZE(512)-
               FILE(BSP3) VOLUMES(333333))
```

Abb. 10.15. Beispiele für die Definition von Haufen

Zu den in Abb. 10.15. dargestellten Beispielen ist folgendes zu sagen:

a) ESDS.A ist mit einem Speicherraum von drei Zylindern versehen.

b) KSDS.B belegt Platz auf zwei Plattenstapeln, die Daten auf dem einen, die Schlüssel auf dem anderen. Die Schlüssel belegen jeweils die ersten drei Bytes der Sätze. Die Kontrollintervalle sind 4096 Bytes lang für die Datensätze und 512 Bytes für die Indexsätze.

Um Haufen oder Speicherräume zu zerstören, haben wir das folgende Statement niederzuschreiben:

```
DELETE   (name [name]···)
         [FILE(dname)]
         [NOPURGE]
         [PURGE   ]
         {CLUSTER}
         {SPACE  }
```

Wird nur ein Name notiert, so können die Klammern entfallen. Um eine Datenmenge mit UNIQUE zu zerstören, wird FILE benötigt und unter name muß der Namenseintrag im VSAM-Katalog erscheinen. Zur Zerstörung eines Speicherraumes wird ebenfalls FILE benötigt, unter name ist jedoch die Datenträgerkennung (Reihennummer) aufzuführen. Um eine Datenmenge zu zerstören, die mit SUBALLOCATION erklärt ist, wird FILE nicht benötigt. PURGE besagt, daß der Haufen oder der Speicherraum auch dann zerstört werden soll, wenn das Verfallsdatum noch nicht erreicht ist. Durch NOPURGE wird hingegen festgelegt, daß bei Nichtüberschreitung des Verfallsdatums die Zerstörung unterbleiben soll. Haufen und Speicherräume können nicht im gleichen DELETE-Statement zerstört werden. Nur leere Speicherräume können der Zerstörung unterzogen werden. Beispiele hierfür sind in der Abb. 10.16. aufgeführt.

```
DELETE    ESDS.A   CLUSTER
DELETE    KSDS.B   FILE(BSP4)PURGE CLUSTER   } ①
DELETE    222222   FILE(BEIS) SPACE
```

Zugehörige Steuerstatements der Jobsteuersprachen in den Betriebssystemen

DOS

```
// DLBL BSP4,'KSDS.B',,VSAM
// EXTENT SYSØØ7,222222
// EXTENT SYSØØ8,333333
// DLBL BEIS,'BEISP',,VSAM
// EXTENT SYSØØ7,222222
// EXEC IDCAMS,SIZE=1ØØK
      ——— ① ———
/*
/&
```

OS

```
//S EXEC PGM=IDCAMS
//BSP4 DD UNIT=333Ø,DISP=OLD,VOL=SER=(222222,333333)
//BEIS DD UNIT=333Ø,DISP=OLD,VOL=SER=222222
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
      ——— ① ———
/*
```

Abb. 10.16. Beispiele für das DELETE-Statement und die dazugehörigen Statements der Jobsteuersprachen der Betriebssysteme DOS und OS

Das Statement

```
PRINT INFILE(dname) [HEX      ]
                    [CHARACTER]
                    [DUMP     ]
```

bewirkt, daß der spezifizierte Haufen entweder in sedezimaler Form (HEX) oder in Zeichenform (CHARACTER) oder in Speicherauszugsform (DUMP, d. h. HEX und

CHARACTER) gedruckt wird, Standardform ist DUMP. Durch die Jobsteuersprachen ist der Name im DLBL- bzw. im DD-Statement festzuhalten; es handelt sich hierbei um den Namen im VSAM-Katalog.

Bei einer Ausführung des IDCAMS-Programmes können beliebig viele DEFINE-, DELETE- und PRINT-Statements verarbeitet werden.

Beim Betriebssystem OS ist es erforderlich, daß der VSAM-Hauptkatalog den Namen AMASTCAT besitzt; das Betriebssystem OS gestattet die Verwendung eines beliebigen, vom Systemprogrammierer festgelegten Namens. Das OS erlaubt die Einrichtung von Benutzerkatalogen. Deshalb kann die Angabe

```
CATALOG(katalogname)
```

den DEFINE- oder DELETE-Statements hinzugefügt werden, um den speziellen Katalog ansprechen zu können. In Programmen wird stets der Hauptkatalog angesprochen, es sei denn, daß ein Statement

```
STEPCAT DD
```

unmittelbar auf das EXEC-Statement folgt oder ein Statement

```
JOBCAT DD
```

unmittelbar auf das JOB-Statement. Werden beide Statements aufgeführt, so gilt für den Step selbstverständlich das Statement mit dem Vermerk STEPCAT. Die meisten OS-Installationen benutzen für Testzwecke Benutzerkataloge.

Aufgabengruppe 25

1) Auf einem Plattenspeicher des Typs IBM 3330 soll eine Datenmenge in der konsekutiven Organisationsform eingerichtet werden, die 100 000 Sätze zu je 100 Bytes umfaßt. Wie viele Zylinder müssen bereitgestellt werden, wenn die Sätze

   a) ungeblockt,
   b) geblockt mit einer Blocklänge von 2000 Bytes,
   c) geblockt mit einer Blocklänge von 4000 Bytes

   gespeichert werden.

2) Auf einem Plattenspeicher des Typs IBM 3330 soll eine Datenmenge indexsequentiell gespeichert werden. Sie besteht aus 100 000 Sätzen, in die jeweils ein sechsstelliger Schlüssel eingebettet ist. Pro Zylinder sind vier Spuren für Überläufe zu reservieren. Wieviele Zylinder sind bereitzustellen, wenn folgende Blocklängen geplant sind?
   a) 100 Bytes (ungeblockt)
   b) 2000 Bytes
   c) 1900 Bytes
   d) 6300 Bytes
   e) 4000 Bytes

3) Die gleiche Datenmenge wie bei Aufgabe 2) ist nach der virtuell sequentiellen Organisationsform einzurichten. Als Freispeicher innerhalb der Kontrollintervalle sind 21 % der Intervallänge vorzusehen. Wieviele Zylinder werden für die Daten und wieviele für die Indizes benötigt, wenn eine Indexspur pro Datenzylinder vorgesehen wird? Die Länge der Kontrollintervalle soll

   a) 4096
   b) 2048
   c) 1024

   für den Datenteil betragen.

4) Von der bei Aufgabe 2) beschriebenen Datenmenge sollen die Inhalte der Schlüsselfelder von jedem Satz auf der Spur 0 des ersten Zylinders herausgeschrieben werden, wenn deren Schlüssel gleich 000002, 000004, 000006, 000008 usw. sind.

5) Besonders wichtig ist die wahlfreie Organisationsform von Datenmengen, wenn mit umfangreichen Matrizen zu arbeiten ist. Wir wollen ein Programm betrachten, das die Elemente a_{ij} einer Matrix von Lochkarten einliest. Die Lochkarten sind dabei wie folgt abgelocht:

Spalten 1 und 2: i
Spalten 3 und 4: j
Spalten 5 bis 12: a_{ij}

Für die Indizes der Matrixelemente gilt $i < 12$ und $j < 4$, so daß höchstens 33 Elemente vorliegen. Nehmen wir an, daß die erste gelesene Karte die Maximalwerte von i und j enthält und daß die folgenden $i \cdot j$ Karten die Matrixelemente a_{ij} zusammen mit ihren Indizes enthalten. Dabei liegen die Matrixelemente in einer willkürlichen Reihenfolge vor. Jedes gelesene Element ist in eine wahlfrei organisierte Datenmenge zu stellen. Nach dem Lesen des letzten Elementes sind die Matrixelemente über einen Zeilendrucker in der Reihenfolge

$$a_{11}, a_{12}, \cdots, a_{i,j-1}, a_{ij}$$

zu drucken. Diskutiere über den Aufbau der wahlfrei organisierten Datenmenge, in die die Matrixelemente hineingestellt werden sollen.

6) Löse die Aufgabe 5) mit einer virtuell sequentiell organisierten Datenmenge, einmal ohne und einmal mit Schlüsseln.

11. Einführung in PL/1

11.1. PL/1

PL/1 ist die Abkürzung von Programming Language/One [1]. In der Zeit von 1963 bis 1966 entwarf eine Arbeitsgemeinschaft, die sich hauptsächlich aus Benutzern von Großcomputern und von IBM-Vertretern zusammensetzte, eine künstliche Sprache, die als PL/1 bekannt wurde. Diese Sprache sollte das Schreiben von Programmen so einfach wie möglich gestalten. Zwar ist nach wie vor in einem gewissen Umfang der Entwurf von allgemeinen und detaillierten Programmablaufplänen erforderlich, doch kann der Programmierer durch den zweckmäßigen Gebrauch von PL/1 die aufwendige geisttötende Arbeit vermeiden, die beim Schreiben von Programmen in der Maschinensprache notwendig ist.

Schreibt ein Programmierer sein Programm in PL/1, so muß er sich nach den Regeln richten, die wir in diesem und den folgenden Kapiteln entwickeln werden. Das geschriebene Programm wird dann abgelocht [2]. Danach wird der Computer mit einem bestimmten Bibliotheksprogramm geladen, das von der IBM zur Verfügung gestellt wird. Von diesem Bibliotheksprogramm wird das PL/1-Programm gelesen, d.h. es wird so behandelt, als ob es Daten wären. Der Computer erzeugt daraus als Ausgabe eine Liste, auf der das eingegebene PL/1-Programm verzeichnet ist, sowie ein äquivalentes Programm in der Maschinensprache. Dieses Bibliotheksprogramm wird PL/1-Kompilierer genannt.

Das vom Programmierer geschriebene PL/1-Programm heißt Quellenprogramm. Es darf nur aus Statements in der PL/1-Sprache bestehen. Das Programm wird normalerweise anschließend in Karten gelocht [3], wobei sinnvollerweise nur die Spalten 2 bis 72 der Karten verwendet werden sollten. In der Regel bleibt die Spalte 1 der Karte frei, um eine Verwechslung zwischen den Steuerstatements der Jobsteuersprache für das Betriebssystem und den PL/1-Statements auszuschalten. Die Spalten 73 bis 80 werden gewöhnlich zur Kennzeichnung des Programmes und zur fortlaufenden Numerierung der Karten verwendet, damit diese wieder sortiert werden können, falls sie einmal unbeabsichtigt fallengelassen werden. Die Eingabe wird als kontinuierlicher Zeichenstrom angesehen, wobei die Spalte 2 einer Lochkarte sich unmittelbar an die Spalte 72 der vorhergehenden Lochkarte anschließt.

Der Arbeitsgang, der darin besteht, das Quellenprogramm in den Computer einzulesen und daraus ein Programm in der Maschinensprache zu erzeugen, heißt Übersetzung (Kompilierung). Das vom Kompilierer erzeugte Programm wird Objektprogramm genannt. Neben dem Objektprogramm gibt der Kompilierer außerdem verschiedene Listen und Diagnostik-Nachrichten aus. Es muß jedoch unter allen Umständen festgehalten werden, daß das Objektprogramm in logischem Sinne völlig identisch dem Quellenprogramm ist, so daß genau die vorgesehene Verarbeitung der Daten bewirkt wird.

Dieser Übersetzungsvorgang entspricht völlig der Übersetzung eines Textes von einer Umgangssprache in eine andere. Jedoch ist im Gegensatz zu den Umgangssprachen die Übersetzung von PL/1 in die Maschinensprache der IBM /370 völlig exakt. Das

---

1) Programmiersprache / 1

2) Die Ablochung kann auf Lochkarten oder Lochstreifen erfolgen; auch eine Direkteingabe ist bei entsprechender Maschinen- und Programmausrüstung, z.B. TSO (Time Sharing Option) möglich.

3) Es ist auch eine Ablochung auf Lochstreifen möglich.

Quellenprogramm muß sich grundsätzlich den strengen Regeln von PL/1 fügen und das Objektprogramm erweist sich als gültiges Programm in der Maschinensprache. Bei der Übersetzung von einer Umgangssprache in eine andere, z.B. vom Englischen ins Deutsche, ist dem Übersetzer eine gewisse Freiheit erlaubt, damit er auf diese Weise einen schlechten deutschen Text vermeiden kann. Im Gegensatz zu menschlichen Übersetzern kann der Computer auch nicht den geringsten Fehler akzeptieren.

Abb. 11.1. zeigt ein kurzes PL/1-Programm. Zur Eingabe gelangt ein jährlicher Zinsfuß, beispielsweise von R=0.0400. Ausgegeben wird eine Tabelle der Beträge, auf die ein Kapital von 1.- DM unter Zugrundelegung des Zinsfußes R durch Zinsen anwächst, wenn diese jährlich dem Kapital zugerechnet werden. Die Tabelle ist beschränkt auf den Zeitraum von N=1 bis N=19 Jahre.

Die erste Zeile enthält einen Kommentar; dieser wird vom Kompilierer nicht verarbeitet. Durch das zweite Statement wird dem Programm ein Namen, nämlich ZINSBER gegeben. Dem nächsten Statement ist ebenfalls ein Name, nämlich P zugewiesen. Das Statement GET LIST(R) bedeutet, daß das Objektprogramm über eine Standardeingabeeinheit, z.B. einem Kartenleser, einen Wert für jede Variable einlesen soll, die in der von Klammern eingeschlossenen Liste steht. Hier tritt nur eine Variable auf, nämlich die Variable R. Die nächsten Statements setzen A auf das Anfangskapital von 1.- DM, die Jahreszahl N auf 1 und die Zinsen F auf 1+R/N. Das Statement PUT LIST(R) druckt lediglich den Wert von R über eine Standardausgabeeinheit, gewöhnlich dem Zeilen- oder Schnelldrucker, aus.

Das nachfolgende Statement ist ebenfalls benannt, sein Name ist Q. Es bewirkt, daß der Wert von A mit dem Wert von F multipliziert wird und das Resultat den vorhergehenden Wert von A ersetzt. Zur Kennzeichnung der Multiplikation benutzen wir ein Sternzeichen, da es kein spezielles Multiplikationszeichen gibt. Das Statement N=N+1 verursacht, daß 1 zum augenblicklichen Wert von N addiert wird und daß die Summe den vorhergehenden Wert von N ersetzt. Man beachte hierbei, daß das Gleichheitszeichen in PL/1 eine andere Bedeutung als das mathematische Gleichheitszeichen besitzt. Das nächstfolgende Statement bewirkt, daß die Programmablaufsteuerung zurück zum Statement Q verzweigt, wenn N kleiner als 20 ist; im anderen Fall erfolgt eine Verzweigung zu P. Wenn die Steuerung zurück nach P geht, so wird ein anderer Wert für R eingelesen und anschließend in gleicher Weise verarbeitet.

Das letzte Statement dient dazu, dem Kompilierer mitzuteilen, daß er das Ende des Programmes erreicht hat. Die Semikolons werden in PL/1 dazu verwendet, um die einzelnen Statements voneinander zu trennen.

```
/*BEISPIEL FUER EINE ZINSBERECHNUNG*/
ZINSBER:   PROCEDURE OPTIONS(MAIN) ;
      P:     GET LIST(R) ;
             A=1.ØØØØ ;
             N=1 ;
             F = 1 + R/N ;
             PUT LIST(R) ;
      Q:     A=A*F ;
             PUT LIST (N,A) ;
             N=N+1 ;
             IF  N<2Ø  THEN GO TO Q ;
                       ELSE GO TO P ;
           END ZINSBER ;
```

Abb. 11.1. Beispiel für eine Zinsberechnung

11.2. Grundlagen der Syntax

Wenn man eine neue Sprache erlernen will, so muß man zuerst ihr Alphabet kennenlernen. PL/1 verwendet 60 Zeichen, die in drei Gruppen eingeteilt werden:

Gruppe 1: Es gibt 29 verschiedene alphabetische Zeichen (Buchstaben):
A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, $,@, #

Gruppe 2: Es gibt die 10 Dezimalziffern (Zahlzeichen):
0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Gruppe 3: Es gibt 21 Sonderzeichen:

=	Gleichheits- oder Ergibtzeichen bzw. Zuweisungszeichen
+	Pluszeichen
-	Minuszeichen
*	Sternzeichen (Multiplikationszeichen)
/	Schrägstrich (Divisionszeichen)
(	Linkes Klammerzeichen (Klammer auf)
)	Rechtes Klammerzeichen (Klammer zu)
,	Komma
.	Punkt (Dezimalpunkt)
'	Anführungszeichen (Apostroph)
%	Prozentzeichen
;	Semikolon
:	Doppelpunkt
<	Kleiner-als-Zeichen
>	Größer-als-Zeichen
|	Oderzeichen
&	Undzeichen
¬	Nichtzeichen
_	Unterstreichungszeichen
?	Fragezeichen
b	Leerzeichen oder Zwischenraumzeichen, auch Blank genannt [1]

Der oft gebrauchte Begriff alphanumerische oder alphamerische Zeichen schließt sowohl Buchstaben als auch Ziffern ein. Die Festlegung von 29 Buchstaben gestattet praktisch die Verwendung verschiedener Alphabete.

Einige Zeichenkombinationen haben eine bestimmte Bedeutung. Dazu zählen:

**	Potenzierung (5 2 → 25)
||	Verkettung ('FD' ||'H' → 'FDH')
>=	größer als oder gleich
<=	kleiner als oder gleich
=	nicht gleich
>	nicht größer als
<	nicht kleiner als
/*	Beginn eines Kommentars
*/	Ende eines Kommentars
→	Pfeil (Zeiger auf)

Da einige Locher und Zeilendrucker nicht für die Verwendung des soeben vorgestellten Zeichenvorrates von 60 Zeichen eingerichtet sind, gibt es einen zweiten, eingeschränkten Zeichenvorrat von 48 Zeichen. Ein Programm muß stets mit einem und nur mit

1) Das Leerzeichen wird zur Vermeidung von Mißverständnissen oft durch ein b gekennzeichnet (Abkürzung von blank)

einem Zeichenvorrat abgefaßt werden; ein Wechsel der Zeichenvorräte mitten in einem Programm ist also nicht gestattet.

Im 48-Zeichenvorrat gibt es folgende Zeichen nicht:

_ @ # % ; : < > | & ¬ ?

An Stelle dieser Zeichen sind im Bedarfsfall in den Programmen bedeutungsgleiche Zeichenkombinationen einzusetzen. In Abb. 11.2. finden wir eine Übersicht über diese Kombinationen.

60-Zeichen-vorrat	48-Zeichenvorrat
:	..
;	,.
%	//
>	GT
<	LT
>=	GE
¬=	NE
<=	LE
¬>	NG
¬<	NL
¬	NOT
&	OR
||	AND
->	CAT
	PT

Abb. 11.2. Ersetzung von Zeichen und Zeichenkombinationen des 60-Zeichenvorrates durch Zeichenkombinationen des 48-Zeichenvorrates

Wenn der 48-Zeichenvorrat benutzt wird, so gelten die Zeichenkombinationen GT, LT usw. als reservierte Wörter, d.h. sie können nirgends in einem anderen Zusammenhang gebraucht werden als in dem in der Abb. 11.2. vorgestellten speziellen Sinn. Im 60-Zeichenvorrat gibt es keine reservierten Wörter.

Eine externe Bezeichnung besteht aus ein bis sieben alphanumerischen Zeichen [1]; das erste muß ein Buchstabe sein. Externe Bezeichnungen sind Namen, die sowohl dem Betriebssystem als auch PL/1-Programmen bekannt sind. Interne Bezeichnungen hingegen können aus 1 bis 31 Zeichen bestehen. Das erste Zeichen muß ein Buchstabe sein, das letzte ein alphanumerisches Zeichen; der Rest darf entweder ein alphanumerisches Zeichen oder ein Unterstreichungszeichen sein. Kein anderes Sonderzeichen außer dem soeben erwähnten Unterstreichungszeichen darf zur Namensbildung herangezogen werden. Ganz klar ist damit auch gesagt, daß inmitten eines Namens keine Leerzeichen erscheinen dürfen. Eine interne Bezeichnung ist ein Name, der innerhalb eines PL/1-Programmes bekannt ist und verwendet wird; außerhalb von PL/1-Programmen ist er unbekannt.

1) Abhängigkeit von den Betriebssystemen; sprachlich ist diese Einschränkung nicht definiert.

Für gültige Bezeichnungen folgen nun einige Beispiele:

```
MARIANNE
ZINS_BER
XYZ
P
A
U17
S
@17
E3RF_9_PU_T
KUNDEN_NAME_UND_ADRESSE
SEXAPPEAL
```

Für ungültige Bezeichnungen folgen ebenfalls einige Beispiele:

```
4F8      (das erste Zeichen ist kein alphabetisches Zeichen)
R$6.4    (diese Bezeichnung enthält ein Sonderzeichen)
ZU TH    (diese Bezeichnung enthält ein Leerzeichen)
```

Wenn eine externe Bezeichnung niedergeschrieben wird, die aus mehr als sieben Zeichen besteht, so bildet der Kompilierer von sich aus eine neue externe Bezeichnung. Diese setzt sich aus den ersten vier und den letzten drei Zeichen der ursprünglich vorgegebenen Bezeichnung zusammen. Doch vergessen wir diese Randbemerkung möglichst schnell: Es ist wirklich kein Merkmal einer guten Programmierung, wenn externe Bezeichnungen länger als sieben Zeichen gemacht werden.

Kommentare dürfen überall dort in ein Programm eingefügt werden, wo auch ein Leerzeichen stehen kann. Alle Kommentare müssen in der Form

```
/*kommentar*/
```

auftreten. Alle Kommentare werden vom Kompilierer nicht beachtet, jedoch erscheinen sie in allen Auflistungen des betreffenden Programmes. Ihre Benutzung in einem Programm ist freigestellt, doch sollten sie oft verwendet werden. Ein Programm wird durch sie übersichtlicher und lesbarer und damit auch verständlicher.

11.3. Problemdaten

Damit der Kompilierer die zur Verarbeitung der Daten erforderlichen Instruktionen aufbauen kann, muß er die Form der zu verarbeitenden Daten kennen. Verschiedene Datentypen können verwendet werden. In diesem Abschnitt wollen wir einige dieser Typen kennenlernen.

Wenn die Daten im standardmäßigen EBCDI-Code [1] dargestellt werden, d.h. ein Zeichen in einem Byte, und dabei so viele Bytes verwendet werden, wie erforderlich sind, so sprechen wir von Zeichenketten. Im Quellenprogramm werden Zeichenketten in der Form

```
'zeichen'
```

geschrieben.

1) EBCDI-Code ist die Abkürzung von Extended Binary coded Decimal interchange Code

Mit Ausnahme des Anführungszeichens dürfen dabei irgendwelche beliebigen Zeichen [1] in Anführungszeichen eingeschlossen werden. Um ein Anführungszeichen in einer Zeichenkette selbst darzustellen, müssen wir das Anführungszeichen zweimal niederschreiben, damit der Kompilierer es von dem Anführungszeichen unterscheiden kann, das das Ende der Zeichenkette angibt. Beispiele für Zeichenketten sind:

```
'ABCDEF'
'S=%.67'
'WEIL''S'
```

Diese stellen der Reihe nach die Zeichenfolgen

```
ABCDEF
S=%.67
WEIL'S
```

dar.

Eine Kette kann wiederholt werden, indem man einen in Klammern eingeschlossenen Wiederholungsfaktor vor das einleitende Anführungszeichen schreibt:

```
(2) 'ABC'
(3) '56'
(0) 'AC%§)'
```

Durch diese Schreibweise werden folgende Ketten dargestellt:

```
ABCABC
565656
(Null)
```

Eine Zeichenkette, welche nur aus zwei Anführungszeichen

```
''
```

besteht oder der der Wiederholungsfaktor 0 vorangeht, wird als Nullkette bezeichnet, weil sie keine Zeichen enthält.

Mitunter ist es notwendig, eine Kette festzulegen, die nur aus Dualziffern besteht. Derartige Bitketten werden wie Zeichenketten niedergeschrieben; sie dürfen jedoch nur die Ziffern 0 und 1 enthalten. Außerdem ist nach dem die Kette abschließenden Anführungszeichen der Buchstabe B zu notieren, um damit diese Kette als Bitkette auszuweisen. Es folgen einige Beispiele für Bitketten:

```
'ØØØØ1Ø1Ø'B
'1111ØØØØ'B
'111111'B
(6)'1'B
```

Innerhalb von Ketten dürfen keine Kommentare niedergeschrieben werden, da der Kompilierer alles, was zwischen den beiden eine Kette begrenzenden Anführungszeichen erscheint, als Teil der Kette ansieht.

---

1) Es sind also alle gültigen Verschlüsselungen des EBCDI-Codes erlaubt, d. h. nicht nur die Verschlüsselungen, die den Zeichen der Zeichenvorräte von PL/1 entsprechen

Bitketten können ausgerichtet oder nicht ausgerichtet angeordnet werden. Die Bits einer ausgerichteten Bitkette werden in aufeinanderfolgenden Bitstellen gespeichert, die an einer Bytegrenze beginnen. Wird nur ein Teil eines Bytes benötigt, so werden nur die linksbündigen Bitstellen zur Speicherung der Bits der Kette herangezogen. Auf diese Weise wird 'ØØ11Ø'B zu ØØ11 ØØØØ (sedezimal also 3Ø) gespeichert; die Kette (6)'1'B zu 1111 11ØØ (sedezimal: FC). Die Bitkette '1111ØØØØ1Ø1Ø'B belegt somit zwei aufeinanderfolgende Bytes und wird zu 1111 ØØØØ 1Ø1Ø ØØØØ (sedezimal: FØAØ) gespeichert. Eine nicht ausgerichtete Bitkette wird nicht automatisch an eine Bytegrenze angepaßt, sie startet vielmehr bei der ersten verfügbaren Bitstelle. Wenn freilich nur eine einzige Bitkette innerhalb eines Hauptspeichergebietes vorgesehen ist, so bleibt nicht anderes übrig, als daß sie bei einer Bytegrenze anfängt. Wenn jedoch weitere Bitketten sich an eine erstdefinierte anschließen, so gibt es doch einen Unterschied gegenüber ausgerichteten Bitketten. Betrachten wir zur endgültigen Klärung dieser Angelegenheit ein Beispiel. Der Bitkette 'ØØ11Ø'B folgt die Bitkette (6)'11'B, wobei die letztere als nicht ausgerichtet deklariert ist. Für beide Bitketten werden insgesamt 17 Bitstellen benötigt, die zu drei aufeinanderfolgenden Bytes gehören. Für diese Bytes ergibt sich die in Abb. 11.3. dargestellte Bitfolge.

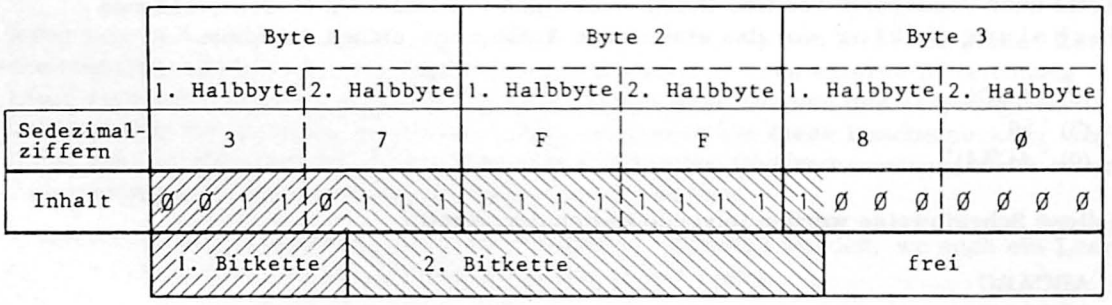


Abb. 11.3. Nicht ausgerichtete Bitketten

Neben den Kettendaten kenne wir auch arithmetische Daten. PL/1 gestattet die Verwendung reeller und komplexer dualer und dezimaler Daten genau so wie die Verwendung von Festpunkt- und Gleitpunktdaten. Alle Daten können dabei mit verschiedenen Genauigkeiten dargestellt werden. Die Genauigkeit einer Festpunktzahl ist definiert durch Angabe der Anzahl ihrer Ziffern sowie durch Angabe der Anzahl ihrer Stellen nach dem Komma. Die Genauigkeit einer Gleitpunktzahl ist definiert durch Angabe der Anzahl ihrer gültigen Ziffern [1], d.h. hier der Anzahl ihrer Mantisssenstellen.

Es folgen als Beispiele einige reelle Festpunktkonstanten mit der Basis 10:

```
72.398
796
784.
.ØØ243
Ø.Ø243
```

Wenn kein Dezimalpunkt geschrieben wird, dann handelt es sich um eine dezimale ganzzahlige Konstante. Reelle dezimale Festpunktzahlen werden in der IBM /370 in Dezimalfeldern in gepackter Form gespeichert, wobei nur so viele Bytes verwendet werden, wie nötig sind. Die Zahl 72.398 wird somit zu 72 39 8+ und die Zahl .00243 zu 24 3+ gespeichert. Der Kompilierer richtet sich bei der Übersetzung nach den Dezimalpunkten, doch speichert er die Dezimalpunkte nicht im gepackten Feld.

1) Unter den gültigen Ziffern einer Zahl versteht man die Ziffern ihrer Ziffernfolge, die nach Streichung der führenden Nullen übrig bleiben.

Es folgen als Beispiele einige reelle Festpunktkonstanten mit dualer Basis:

```
11111ØØ11Ø.ØØØ1B
1ØØ1Ø11ØB
1ØØØ1ØØB
.ØØØØØ1ØØ1B
```

Beachte hierbei, daß der Buchstabe B eingesetzt wird, um die dualen Konstanten von den dezimalen Konstanten zu unterscheiden. Duale Werte werden in der IBM /370 als Dualzahl in einem vollen Wort gespeichert, wenn sie mehr als 15 Dualziffern aufweisen. Sonst erfolgt ihre Speicherung in einem Halbwort.

Es folgen nunmehr einige reelle dezimale Gleitpunktkonstanten:

```
234.5E+7
236E-2
6.78E2
Ø.234E+Ø2
.ØØØ45E21
```

Die Notierung des Buchstabens E kennzeichnet derartige Konstanten als Gleitpunktkonstanten. Um den Wert derartiger Konstanten zu bestimmen, muß man die vor dem E stehende Zahl mit der Potenz multiplizieren, deren Basis gleich 1Ø ist und deren Exponent gleich der Zahl ist, die auf den Buchstaben E folgt. Somit bedeutet 234.5E+7 den Wert, der sich aus $234{,}5 \cdot 10^7$ ergibt, nämlich 2 345 000 000. Für 236E-2 ergibt sich demnach 2,36. Dezimale Gleitpunktkonstanten mit sechs oder weniger gültigen Ziffern vor dem Buchstaben E werden in der IBM /370 als Gleitpunktzahlen in einem vollen Wort gespeichert; Gleitpunktkonstanten mit mehr als sechs gültigen Ziffern belegen ein Doppelwort, sofern sie nicht mehr als 16 Ziffern aufweisen. Alle anderen werden in zwei Doppelwörtern als Gleitpunktzahlen mit erweiterter Genauigkeit gespeichert [1].

Es sollen nunmehr noch reelle duale Gleitpunktkonstanten besprochen werden. Einige Beispiele hierzu sind:

```
1.11Ø1E3B
.ØØ11Ø1E-4B
Ø.11ØØ1E6B
```

In diesem Fall bedeutet die auf den Buchstaben E folgende Zahl einen Exponenten zur Basis 2. Somit besitzt die Konstante 1.1101E3B den Wert $1{,}1101 \cdot 2^3 = 1110{,}1_2$. Duale Gleitpunktkonstanten werden in der IBM /370 als Gleitpunktzahlen in einem vollen Wort gespeichert, wenn sie 21 oder weniger gültige Ziffern aufweisen; anderenfalls belegen sie ein Doppelwort, sofern sie weniger als 54 Ziffern links vor dem Buchstaben E besitzen. Alle anderen werden in zwei Doppelwörtern als Gleitpunktzahlen mit erweiterter Genauigkeit gespeichert. [1]

Eine imaginäre Konstante wird wie eine reelle Konstante geschrieben, nur daß ihr außerdem der Buchstabe I angehängt wird. Daraus folgt, daß

3.21I	eine imaginäre dezimale Festpunktkonstante,
1ØØ11BI	eine imaginäre duale Festpunktkonstante,
1.2E+2I	eine imaginäre dezimale Gleitpunktkonstante und
Ø.ØØ1Ø1ØE+3BI	eine imaginäre duale Gleitpunktkonstante

ist.

1) Die Kompilierer, die ins Betriebssystem DOS eingebettet sind, gestatten nicht die Verwendung von Gleitpunktzahlen mit erweiterter Genauigkeit.

Eine komplexe Konstante wird in PL/1 einfach als Summe einer reellen und einer imaginären Konstante niedergeschrieben. Somit bedeuten

```
56.7+3.27I
3.2E+Ø-2.7E-1I
1ØØ1B+11ØØBI
```

komplexe Konstanten. Der Imaginärteil derselben ist am angehängten Buchstaben I zu erkennen. Im System IBM /370 werden komplexe Konstanten als ein Paar von reellen Konstanten gespeichert.

11.4. Konstanten und Variablen

Um eine konstante Größe festzuhalten, schreiben wir einfach diese Konstante nieder. Wollen wir uns dagegen auf eine Größe beziehen, deren Wert wir nicht kennen, so müssen wir für diese Größe eine Bezeichnung niederschreiben. Für eine Variable kann irgendeine gültige Bezeichnung verwendet werden; einzige Ausnahme: bei Verwendung der 48-Zeichen-Menge dürfen die reservierten Wörter nicht benutzt werden, um Variable zu bezeichnen.

Werden keine anderslautenden Vereinbarungen vorgenommen, so nimmt der Kompilierer an, daß der Wertevorrat von Variablen, deren Namen mit I, J, K, L, M und N beginnen, reelle duale Festpunktzahlen mit der Genauigkeit (15,Ø) sind, d.h. Zahlen mit 15 Dualziffern und keinem gebrochenen Anteil. Der Wertevorrat von Variablen, deren Namen mit einem anderen alphabetischen Zeichen beginnen, wird dagegen zu reellen Gleitpunktzaheln mit der Genauigkeit (6) angenommen, d.h. mit sechs gültigen Dezimalziffern. Beispiele für diese Standardannahmen stellen die folgenden Variablennamen dar:

IKY	duale Festpunktvariable
WKY	dezimale Gleitpunktvariable
@78	dezimale Gleitpunktvariable

Alle drei Variablen können nur reelle Werte annehmen.

Mitunter entsprechen diese Annahmen nicht unseren Vorstellungen. Der Programmierer kann sich über diese den Kompilierer betreffenden impliziten Annahmen hinwegsetzen, indem er in einem DECLARE-Statement die von ihm gewünschten Eigenschaften angibt. Dieses Statement besitzt die Form:

```
DECLARE  name attribute,
         name attribute,
         ...
         name attribute ;
```

Zur Verbesserung der Lesbarkeit können Leerzeichen verwendet werden: Zwischen zwei Wörtern muß mindestens ein Leerzeichen stehen [1], weitere Leerzeichen werden vom Kompilierer nicht beachtet. In einem DECLARE-Statement können die Attribute so vieler Variabler erklärt werden, wie man nur immer wünscht. Somit können wir z.B. schreiben:

```
DECLARE  KMY FIXED(5,3),
         IU79 BINARY(7,2),
         X BINARY FLOAT REAL,
         Y CHARACTER(7) ;
```

1) Ein Leerzeichen ist dann nicht erforderlich, wenn die Wörter durch ein anderes Trennungszeichen getrennt werden.

Werden für eine Variable keine Attribute niedergeschrieben, so ordnet der Kompilierer dieser Variablen die standardmäßigen Attribute implizit zu, so wie sie weiter oben bereits angegeben wurden.

Für Kettendaten gibt es die Attribute BIT bzw. CHARACTER sowie das Längenattribut, das in Klammern eingeschlossen werden muß. (Das Attribut CHARACTER darf mit CHAR abgekürzt werden). Die im obigen Beispiel aufgeführte Zeile

```
Y CHARACTER(7)
```

bedeutet somit ein aus sieben Bytes bestehendes Feld, das den Wert der Zeichenkettenvariablen mit dem Namen Y aufnimmt. Die Schreibweise

```
Y CHAR(7)
```

ist der bereits besprochenen Schreibweise völlig äquivalent. In gleicher Weise definiert

```
B BIT(16)
```

eine Variable mit dem Namen B, deren Wertevorrat sich aus Bitketten zusammensetzt, die sechzehn Bitstellen (gleich zwei Bytes) belegen. [1]

Wenn keine Länge für Kettenvariablen angegeben ist, so wird ihnen die Länge 1 unterstellt. Somit ist

```
T BIT    äquivalent    T BIT(1)
```

Für arithmetische Daten gibt es folgende Attribute:

a) DECIMAL oder BINARY zur Kennzeichnung der Basis ; wird keines von beiden angegeben, so wird standardmäßig das Attribut DECIMAL zugeordnet.

b) FIXED oder FLOAT als Skalierungsattribute; würde keines von beiden niedergeschrieben, so wird standardmäßig das Attribut FLOAT zugeordnet.

c) REAL oder COMPLEX als Modusattribute; wird keines von beiden niedergeschrieben, so wird standardmäßig das Attribut REAL zugeordnet.

Die Genauigkeit wird für Festpunktdaten in der Form (w,d) und für Gleitpunktdaten in der Form (w) angezeigt. Durch w wird die Gesamtzahl der Ziffern angegeben, die für einen Wert, den die betreffende Variable annehmen kann, erlaubt ist (bei Gleitpunktdaten gibt w die maximale Anzahl der erlaubten gültigen Ziffern an). Durch d wird die Anzahl der nach dem Dual- bzw. Dezimalpunkt stehenden Ziffern angegeben. Wird auf die Niederschrift von d verzichtet, so nimmt der Kompilierer an, daß d=0 ist. Komplexe Zahlen werden als ein Paar reeller Zahlen dargestellt; Realteil und Imaginärteil besitzen die gleichen Attribute.

Der Kompilierer ordnet denjenigen Variablen bestimmte Genauigkeitsattribute zu, für die keine angegeben sind. Infolge des logischen Aufbaus des System IBM /370 dürfen andererseits gewisse Grenzen für die Genauigkeitsattribute nicht überschritten werden. Die zugeordneten und die maximal möglichen Genauigkeitsattribute können der Tabelle entnommen werden, die in Abb. 11.4. dargestellt ist. Die zugeordneten Genauigkeitsattribute werden wir zukünftig als Standardgenauigkeitsattribute bezeichnen.

1) Bitketten bzw. Zeichenketten dürfen nicht mehr als 32767 Bitstellen bzw. Bytes belegen; der Kompilierer läßt ein größeres Längenattribut nicht zu.

Attribute	Standardgenauigkeit	Maximalgenauigkeit	Bemerkungen
DECIMAL FIXED	(5,0)	(15,d)	
BINARY FIXED	(15,0)	(31,d)	
DECIMAL FLOAT	(6)	(33)	DOS-Kompilierer: (16)
BINARY FLOAT	(21)	(1Ø9)	DOS-Kompilierer: (53)

Anmerkung: d kann sowohl positive als auch negative ganzzahlige Werte annehmen.

Abb. 11.4. Genauigkeitsattribute arithmetischer Variabler

Die DOS-Kompilierer haben für Gleitpunktdaten eine eingeschränkte Maximalgenauigkeit, da sie keine Gleitpunktzahlen mit erweiterter Genauigkeit zulassen. In der Tabelle von Abb. 11.4. finden sich die entsprechenden Bemerkungen.

Machen wir noch einen kleinen Exkurs. Wir wollen einmal sehen, was ein negativer Wert für d für Konsequenzen hat. Nehmen wir ein Beispiel her:

```
DECLARE  N DECIMAL FIXED(5,-3) ;
```

Eine solche Erklärung besagt, daß die Variable namens N insgesamt fünf Ziffernstellen besitzt. Die Stellung des Dezimalpunktes wird drei Stellen rechts von der letzten Ziffernposition angenommen. Ihr Wertevorrat, absolut gesehen, reicht deshalb von 00000000 bis 99999000. Wenn also das N zugeordnete Feld im Hauptspeicher den Inhalt 33 44 5+ besitzt, so repräsentiert dieser den Wert + 33 445 000 .

Wenn das Genauigkeitsattribut in Erklärungen enthalten ist, so muß es einem der anderen Attribute (DECIMAL, BINARY, FIXED, FLOAT, REAL oder COMPLEX) folgen. Es ist somit unmöglich, in eine Erklärung allein das Genauigkeitsattribut einzubeziehen; wenigstens ein anderes Attribut muß zusätzlich vorhanden sein. Alle anderen Attribute dürfen in einer beliebigen Reihenfolge aufgeführt werden.

Wenn eine Variable aus irgendeinem Grunde auch außerhalb eines Programmes (z. B. in einem anderen Programm) bekannt sein muß oder soll, so ist zusätzlich das Attribut EXTERNAL anzugeben. Fehlt dieses den Gültigkeitsbereich eines Namens beeinflussende Attribut, so wird das Attribut INTERNAL angenommen. Wenn jedoch EXTERNAL niedergeschrieben wird, so sollte die Bezeichnung der Variablen den Erfordernissen der Betriebssysteme für die Namensgebung von externen Größen (siehe Abschnitt 11.2.) genügen.

Die Standardattribute, die wir soeben besprochen haben, können durch die DEFAULT-Vereinbarung außer Kraft gesetzt und durch neue Standardattribute ersetzt werden. Das DEFAULT-Statement besitzt folgende allgemeine Form:

```
DEFAULT  RANGE(umfang) attribute,
         RANGE(umfang) attribute,
         :
         RANGE(umfang) attribute ;
```

Unter attribute können beliebige gültige Attribute niedergeschrieben werden. Unter umfang ist der Namenskreis der Variablen zu verstehen, für die die nachfolgenden Attribute gelten sollen . Für umfang können drei verschiedene Formen niedergeschrieben werden:

a) *

Das Sternzeichen bedeutet, daß die Attributmenge auf alle Variablen anzuwenden ist, sofern diese einer Attributzuordnung bedürfen.

b) folge
Die Niederschrift einer Zeichenfolge bedeutet, daß die Attributmenge allen Variablen zugeordnet werden soll, deren Namen mit folge beginnen.

c) abuchstabe:ebuchstabe
Diese Niederschrift bedeutet, daß die neue Standardattributmenge allen Variablen zugeordnet werden soll, deren Namen mit einem Buchstaben beginnt, der zum spezifizierten Buchstabenkreis gehört. Der Buchstabenkreis fängt mit dem Buchstaben abuchstabe an und reicht bis zum Buchstaben ebuchstabe; beide Randbuchstaben sind in den Kreis eingeschlossen.

Sehen wir uns einige Beispiele an :

```
DEFAULT  RANGE(*) BINARY,
         RANGE(A:B) BINARY FIXED,
         RANGE(PQR) VALUE(BINARY FIXED(15,4)) ;
```

Attribute müssen in der Form

```
VALUE(attribute)
```

niedergeschrieben werden, wenn sie konstante Angaben einschließen, wie sie beispielsweise das Genauigkeitsattribut erfordert. Die oben aufgeführte Einrichtung neuer Standardattribute hat zur Folge, daß allen Variablen das Standardattribut BINARY zukommt, sofern ihnen nicht explizit in einer DECLARE-Vereinbarung ein anderes zugeordnet wird, dem BINARY widerspricht. Alle Variablen, deren Name mit A, B oder C anfängt, bekommen die Standardattribute BINARY und FIXED, sofern ihnen nicht explizit andere widersprechende Attribute zugeordnet werden. Schließlich werden allen Variablen, deren Name mit PQR beginnt, die Attribute BINARY FIXED(15,4) zugeordnet.

Bei Abwesenheit eines DEFAULT-Statements, das neue Standardattribute festlegt, bedeutet die Angabe irgendeines arithmetischen Attributes, daß an die Stelle fehlender arithmetischer Attribute DECIMAL, FLOAT, oder REAL treten.

Es wird von uns empfohlen, DEFAULT-Statements am Programmanfang zu bringen, doch notwendig ist das nicht. Der Kompilierer akzeptiert sie an jeder beliebigen Programmstelle. Unberücksichtigt des Ortes ihrer Niederschrift gelten sie für alle Variablen innerhalb eines Programmes.

11.5. Schreibweise (Notation)

Beim Studium von Programmiersprachen taucht öfters der Wunsch auf, daß ein formales Statement präsent sei, nach dem man sich beim Niederschreiben eines Statements richten kann. Einige standardisierte Schreibweisen werden allgemein bei allen Programmiersprachen verwendet. Wir sollten uns diesen Formalismen anpassen. Doch stellen wir sie zunächst einmal vor.

1) Schlüsselwörter wie DECLARE werden in Großbuchstaben geschrieben. Solche Wörter werden genau so niedergeschrieben, wie sie im Formalstatement stehen.

2) Bezeichnungen, über deren Vergabe der Programmierer verfügen kann, werden mit Kleinbuchstaben geschrieben. [1)]

3) Wenn zwischen zwei oder mehreren Alternativen gewählt werden kann, so werden diese in geschweifte Klammern gestellt

1) Der Deutlichkeit wegen und um Verwechslungen im erklärenden Text vorzubeugen, sind sie in diesem Buch zusätzlich unterstrichen.

4) Wenn eine Wahl in der Weise getroffen werden muß, daß eine oder keine der angebotenen Alternativen genommen werden muß, so werden die Alternativen in eckige Klammern gesetzt. Z.B. gilt das für die Skalierungsattribute von arithmetischen Variablen

$$\left[\begin{matrix}\text{FIXED}\\ \text{FLOAT}\end{matrix}\right]$$

5) Gibt es bei alternativen Schlüsselwörtern eine Standardzuordnung und tritt diese bei Fehlen einer Alternative ein, so ist der Standard unterstrichen, z.B.

$$\left[\begin{matrix}\underline{\text{FIXED}}\\ \text{FLOAT}\end{matrix}\right]$$

6) Drei aufeinanderfolgende Punkte in Zeilenmitte bedeuten, daß das vorhergehende Element so viele Male wiederholt werden kann, wie es gewünscht wird. Ein einzelnes Element in eckigen Klammern muß den drei Punkten vorangestellt werden, um anzuzeigen, wie das Element aussieht, das wiederholt werden kann.

7) Anstatt die Alternativen übereinander, d.h. in mehrere Zeilen zu schreiben, können diese auch auf einer Zeile erscheinen; sie müssen dann jedoch durch einen Vertikalstrich voneinander getrennt werden. Ein Beispiel hierzu:

[FIXED|FLOAT]

Im allgemeinen können bei PL/1 die Attribute in einer beliebigen Reihefolge geschrieben werden (siehe Abschnitt 11.4.).

Um die obigen Schreibweisen zu demonstrieren, wollen wir die Regeln für das DECLARE-Statement einmal zusammenfassend darstellen.

DECLARE {name attribute} [,···] ;

Unter attribute ist hierbei zu verstehen:

$$\left[\begin{matrix}\underline{\text{REAL}}\\ \text{COMPLEX}\end{matrix}\right]\ \left[\begin{matrix}\underline{\text{DECIMAL}}\\ \text{BINARY}\end{matrix}\right]\ \left[\begin{matrix}\underline{\text{FIXED}}\\ \text{FLOAT}\end{matrix}\right]\ [(\underline{w}\,[,\underline{d}])]\ \left[\begin{matrix}\underline{\text{INTERNAL}}\\ \text{EXTERNAL}\end{matrix}\right]$$

Durch geschweifte Klammern um name und attribute haben wir eine Gruppierung angezeigt. Damit deuten wir an, daß die nachfolgenden drei aufeinanderfolgenden Punkte eine Wiederholung anzeigen, die auf die in den geschweiften Klammern enthaltene Gruppierung bezug nimmt. Man beachte ferner, daß alle Attribute nur Optionen darstellen: Die Attributangaben können also gänzlich fehlen.

11.6. Programmsteuerungsdaten

Im Programm der Abb. 11.1. ist die Variable A implizit als dezimale Gleitpunktvariable mit der Genauigkeit 6 erklärt, desgleichen F und R. N hingegen ist implizit als duale Festpunktvariable mit der Genauigkeit 15 erklärt. Neben diesen Bezeichnungen treten drei andere Bezeichnungen im Programm auf, nämlich ZINSBER, P und Q. Von diesen ist ZINSBER der Name der Prozedur, unter dem das Betriebssystem OS das Programm kennt. ZINSBER ist, vom Programmierer her gesehen, ein *externer Name* in dem Sinn, daß er außerhalb des Programmes bekannt ist (nämlich im Betriebssystem); die anderen beiden Namen sind *intern* zum Programm, d.h. sie sind nur im Programm bekannt. Die beiden Bezeichnungen P und Q werden als *Markenkonstanten* bezeichnet. Sie sind Konstanten, da sie während des Programmablaufes nicht verändert werden. Später werden wir noch andere Arten von

Programmsteuerungsdaten kennenlernen. In diesem Kapitel begegnen wir Markenkonstanten als dem einzigen Typ von Bezeichnungen, die nicht externe Namen oder Problemdatenvariable kennzeichnen.

Die Namen von Marken und Prozeduren gehören zur Gattung der Programmsteuerungsdaten. Später werden wir noch andren Typen von Programmsteuerungsdaten begegnen. Vorerst jedoch beschränken wir uns auf folgende Arten von Daten:

a) Problemdaten: arithmetische Daten, Kettendaten
b) Programmsteuerungsdaten: Markendaten, Prozedurnamen.

11.7. Arithmetische Ausdrücke

Unter einem arithmetischen Ausdruck versteht man irgendeine Kombination von Variablen und Konstanten, die durch Operatoren (+, - usw.) und Klammern derart voneinander getrennt sind, daß sie eine sinnvolle mathematische Formel ergeben. Es folgen einigeBeispiele:

```
Y + 17.45          Y + (.Ø18/Y)          -7 - YU7
```

Folgende Operatoren können verwendet werden:

1) Vorzeichen: + und - (Prefixoperatoren)
2) Potenzieren: * *
3) Multiplizieren und Dividieren: * und /
4) Addieren und Subtrahieren: + und - (Infixoperatoren)

Da in der Mathematik Pluszeichen und Minuszeichen eine doppelte Bedeutung besitzen, nämlich einmal als Vorzeichen positiver und negativer Zahlen, zum anderen als Additions- bzw. Subtraktionszeichen, müssen wir beide Bedeutungen unterscheiden; wir bezeichnen die Zeichen deshalb entweder als Prefix-Operatoren oder als Infix-Operatoren. Als Prefix-Operator wollen wir dabei einen Operator verstehen, dem weder eine Variable noch eine Konstante vorangeht; als Infix-Operator soll dabei ein Operator bezeichnet werden, der in einem Ausdruck einem andren Glied (Term) folgt. Somit stellt im dritten der obigen Beispiele das Minuszeichen vor der 7 einen Prefix-Operator und das Minuszeichen vor YU7 einen Infix-Operator dar. Zwei aufeinanderfolgende Sternzeichen werden als ein einziges Operatorsymbol angesehen und repräsentieren die Potenzierung, wobei der vor diesem Symbol stehende Operand als Basis und der nach diesem Symbol stehende Operand als Exponent interpretiert wird. Beispiele hierfür sind:

$2 ** 3 \triangleq 2^3 = 8$
$3 ** 2 \triangleq 3^2 = 9$

Wenn in einem Ausdruck keine Klammern auftreten, die die Reihenfolge der zur Berechnung des Ausdruckes durchzuführenden Operation festlegen, so werden die Prefixoperationen und die Potenzierungen zuerst ausgeführt. Bei mehreren Operationen dieser Art in einem Ausdruck werden sie von rechts nach links behandelt. Nach Auswertung aller Operationen dieser 1. Rangordnung folgen die Multiplikationen und Divisionen (2. Rangordnung). Daran schließen sich Additionen und Subtraktionen (3. Rangordnung) an. Bei mehreren Operationen gleicher Rangordnung in einem Ausdruck erfolgt deren Ausführung von links nach rechts, also gerade in der umgekehrten Reihenfolge wie bei den Operationen der 1. Rangordnung. Somit wird der Ausdruck

```
-7+.Ø18/Y+3
```

so behandelt, als ob er in der Form

```
((-7)+(.Ø18/Y))+3
```

geschrieben wäre.

Im allgemeinen dürfte daraus kein Problem entstehen, doch sollte man in einigen Fällen vorsichtig sein. Der Ausdruck

```
2**3**2
```

wird interpretiert zu

```
2**(3**2)=2**9
```

wodurch sich der Potenzwert 512 ergibt; der Ausdruck

```
(2**3)**2
```

hingegen wird zu

```
8**2
```

interpretiert; der Potenzwert ist in diesem Falle also gleich 64.

Im Gegensatz zur gewöhnlichen mathematischen Schreibweise schließt die Nebeneinanderstellung zweier Operanden (Konstanten bzw. Variablen) keine Operation ein. Der Ausdruck

```
YU7(3+X)
```

ist also nicht definiert. Soll eine Multiplikation erfolgen, so muß der dafür vorgesehene Operator, das Sternzeichen, unbedingt niedergeschrieben werden, also

```
YU7 * (3+X)
```

Man sollte tunlichst niemals zwei Operatoren nebeneinander schreiben, d.h. zwischen zwei Operatoren sollten stets Klammerzeichen oder Glieder stehen. Der Ausdruck

```
YU7 + 3**-7 + 4
```

z.B. sollte deshalb besser zu

```
YU7 + 3**(-7) + 4
```

geschrieben werden oder entsprechend anders, wenn eine andere Reihenfolge der Berechnungen gemeint ist.

Treten in einem in PL/1 niedergeschriebenen Ausdruck sowohl reelle als auch komplexe Glieder auf, so werden die reellen Glieder in solche komplexen Glieder umgewandelt, deren Imaginärteil gleich Null ist. Treten sowohl Glieder in Festpunktdarstellung als auch in Gleitpunktdarstellung auf, so werden zunächst die Festpunktglieder in Gleitpunktglieder umgewandelt. Werden sowohl dezimale als auch duale Glieder verwendet, so werden zunächst die dezimalen Glieder in duale konvertiert (umgewandelt). Unterscheiden sich die Glieder in ihren Genauigkeiten, so hält der Kompilierer die größere Genauigkeit fest; er kann jedoch keine größere Genauigkeit als diejenige festhalten, die für ein gepacktes Dezimalfeld (15 Dezimalstellen) oder für ein Dualwort (31 Bitstellen) oder für zwei Doppelwörter zur Aufnahme von Gleitpunktzahlen erweiterter Genauigkeit (28 sedezimale Mantissenstellen).

Da also in einem Ausdruck das Auftreten von Gliedern mit ungleichen Attributen erlaubt ist, sollte sich der Programmierer immer vergegenwärtigen, daß der Kompilierer in solchen Fällen in das Objektprogramm Instruktionen einfügen muß, die die erforderlichen Umwandlungen bewirken. Diese zusätzlichen Instruktionen verlangsamen selbstverständlich die Ausführung des Programmes (machen es dynamisch länger) und belegen außerdem zusätzlichen Speicherplatz (machen also das Programm auch statisch länger). Deshalb sollte als gute Programmierregel gelten, so wenig Glieder mit ungleichen Attributen in Ausdrücken zu verwenden wie nur irgendwie möglich ist.

Es folgen einige Beispiele für gültige Ausdrücke:

```
B**2 - 4*A*C
(-B+DELTA) / (A+A)
(A*B-C*D) * (A*C+B*D)
```

Einige spezielle Regeln betreffen die Berechnung von Potenzen:

a) Eine Potenz mit der Basis 0 und dem Exponenten 0 bzw. einem negativen Exponenten ist fehlerhaft und kann nicht berechnet werden.

b) Eine Potenz mit der Basis 0 und einem positiven Exponenten ergibt den Potenzwert 0.

c) Eine Potenz mit einer von 0 verschiedenen Basis und dem Exponenten 0 ergibt den Potenzwert 1.

d) Eine Potenz mit positiver Basis und einem ganzzahligen Exponenten in Festpunktdarstellung wird durch fortlaufende Multiplikation berechnet, bei irgendeinem anderen Exponententyp erfolgt die Berechnung über die Logarithmus- und Exponentialfunktion.

e) Eine Potenz mit negativer Basis und einem ganzzahligen positiven Exponenten in Festpunktdarstellung wird durch fortlaufende Multiplikation berechnet.

f) Eine Potenz mit negativer Basis und einem Exponenten, der nicht ganzzahlig positiv in Festpunktdarstellung ist, ergibt einen Fehler.

g) Ein komplexer Exponent ist dann und nur dann erlaubt, wenn sein Imaginärteil gleich Null ist; ist der Imaginärteil jedoch ungleich Null, so ergibt sich ein Fehler.

h) Eine Potenz mit der komplexen Basis 0 und einem Exponenten mit positivem Realteil und dem Imaginärteil 0 ergibt 0.

Wird eine Bitkette in einem arithmetischen Ausdruck verwendet, so wird sie in eine Dualzahl mit den Attributen BINARY FIXED(15,0) umgewandelt.

Tritt eine Zeichenkette in einem arithmetischen Ausdruck auf, so muß sie eine gültige Konstante darstellen, d.h. sie muß in einer der Formen erscheinen, die für Konstanten mit den Attributen FIXED oder FLOAT, DECIMAL oder BINARY, REAL oder COMPLEX erlaubt sind. Die Zeichenkette wird dann zunächst in denjenigen Konstantentyp umgewandelt, den sie enthält, bevor sie in die arithmetischen Operationen eingeht. Derartige Umwandlungen sind relativ aufwendig und sollten daher tunlichst vermieden werden.

11.8. Ergibtanweisungen

Die Ergibtanweisung (das Ergibtstatement) besitzt die Form:

```
[marke:]  variable=ausdruck ;
```

Der sich bei der Berechnung des Ausdruckes ergebende Wert wird so umgewandelt, daß er den Attributen der links vom Ergibtzeichen (Zuweisungszeichen) stehenden Variablen gerecht wird. Danach wird dieser Wert der Variablen zugewiesen, d.h. er ersetzt den augenblicklichen Wert der Variablen. Nachstehend folgen einige Beispiele von Ergibtanweisungen:

```
        X1 = -B/2 ;
        A = A*F ;
RADDR:  N = N+1 ;
```

Mehr als eine Marke kann durchaus verwendet werden. Jeder Marke muß jedoch ein Doppelpunkt folgen. Beispiel:

```
WDR:  RADDR:  N = N+1 ;
```

Im allgemeinen besteht kein Vorteil in der Verwendung von mehr als einer Marke.

Mitunter wollen wir einen Ausdruck auswerten und seinen Wert mehr als einer Variablen zuweisen. Um dies zu bewerkstelligen, können wir einfach jede Variable links vom Ergibtzeichen niederschreiben, doch müssen wir zwischen die einzelnen Variablen jeweils ein Komma setzen. Somit wird durch die Ergibtanweisung

```
JHR:  X,Y,Z = Ø ;
```

der Wert 0 den drei Variablen X, Y und Z zugewiesen. Diese Ergibtanweisung ist infolgedessen den drei Ergibtanweisungen

```
JHR:  X = Ø ;
      Y = Ø ;
      Z = Ø ;
```

völlig gleichwertig.

In diesem Fall läuft bei der Programmausführung keine arithmetische Operation ab, hier handelt es sich vielmehr um eine einfache Datenübertragung der Zahl 0, die irgendwo im Hauptspeicher gespeichert ist, in die drei verschiedenen Speicherzellen, die die Variablen X, Y und Z repräsentieren.

Aufgabengruppe 26

1) Welche der folgenden Zeichenketten sind gültige PL/1-Bezeichnungen?

   a) DAGMAR b) FRANZBRANNTWEIN c) # 007 d) 7896 e) MAU) f) DOLLARS$
   g) AIIIIIIIIIIIIIS h) B_V_N_M i) /*IFA j) M@7H k) BK_350 l) $453 m) _E_R.

2) Welche der in Aufg. 1) aufgeführten Bezeichnungen kann als Marke verwendet werden?

3) Schreibe die Kette auf, die für die nachfolgenden Kettenkonstanten erzeugt wird.

   a) 'GEHT''S' b) '0010' c) '0010'B d) (4)'J8' e) (5)'00011000'B

4) Wie viele Bytes belegt im Objektprogramm jede der in Aufgabe 3) niedergeschriebenen Kettenkonstanten?

5) Untersuche die folgenden Konstanten, ob sie eine Zeichenkette, eine Bitkette oder eine arithmetische Konstante darstellen! Handelt es sich um arithmetische Konstanten, so ermittle ihre Attribute (Eigenschaften), d.h. REAL oder COMPLEX, FIXED oder FLOAT, DECIMAL oder BINARY.

a) 'ROLLS' b) 'RR' c) 45 d) 0010.09 e) 0010.09E+00 f) 7+4I g) 7E3-2E4I
h) 100010111B i) 2000100020B j) 1001E+4B k) 123456.78E+34 l) '010201'B

6) Wie viele Bytes belegt jede der in Aufg. 5) vorgestellten gültigen Konstanten?

7) Es ist J=4, K=5, und L=3. Welchen Wert nehmen die Variablen auf der linken Seite des Ergibtzeichens an, wenn folgende Ergibtanweisungen ausgeführt werden?

a) NA=J*K; b) NB=J/(-L); c) NC=K* L; d) ND=L**K; e) NE=(((K+K)*J/2)+2)/8;

Es soll vorausgesetzt werden, daß keine der auf der linken Seite der Ergibtzeichen stehenden Variablen in einem DECLARE-Statement erklärt ist.

11.9. Programmablaufsteuerung

Normalerweise beginnt die Ausführung eines PL/1-Programmes mit dem ersten ausführbaren Statement; ein solches Statement soll deshalb als Anweisung bezeichnet werden. (PROCEDURE-, DECLARE-Statements usw. sind keine Anweisungen). Nach der ersten Anweisung werden die nachfolgenden Anweisungen ausgeführt.

Die Vereinbarung

```
name:  PROCEDURE OPTIONS(MAIN) ;
```

muß als erstes Statement jedes PL/1-Programmes codiert werden. Der in diesem Statement aufgeführte Name gilt als externe Bezeichnung. Dem Betriebssystem ist deshalb das Programm unter diesem Namen bekannt. Die Option

```
OPTIONS(MAIN)
```

besagt, daß bei Ausführung des Objektprogrammes die Programmablaufsteuerung vom Betriebssystem unmittelbar der ersten Anweisung dieser Prozedur übertragen wird, d.h. einem Statement, das nicht gleich einer Vereinbarung, wie DECLARE, DEFAULT und PROCEDURE ist.

Die Anweisung

```
{GO TO}
{GOTO } marke ;
```

bewirkt, daß das Programm unbedingt mit derjenigen Anweisung fortgesetzt wird, die durch die angeführte Marke gekennzeichnet ist, auf diese Weise beginnt die Ausführung einer neuen Folge von Anweisungen. Das GO TO - Statement darf selbstverständlich selbst auch durch eine Marke markiert werden. Beispiele für unbedingte Verzweigungsanweisungen sind:

```
        GO TO WDR ;
X7U:    GO TO X8P ;
X8P:    GO TO P88 ;
        GO TO P887 ;
```

Die Anweisung

```
ON bedingung  statement
```

bezieht sich auf eine Bedingung, die auftreten kann, aber nicht muß. Tritt die Bedingung auf, so wird das Statement ausgeführt, anderenfalls nicht. Nur eine einfache Anweisung darf als Statement niedergeschrieben werden. Dieser Anweisung darf außerdem keine Marke vorangehen. Im allgemeinen wird man vorzugsweise ein GOTO-Statement codieren. Es können eine ganze Anzahl von Bedingungen geprüft werden; wir wollen uns zunächst nur mit einer einzigen Bedingung befassen, nämlich mit der Dateiende-Bedingung. Diese Bedingung tritt immer dann auf, wenn bei der angegebenen Eingabedatei das Ende festgestellt wird. Beispiele hierfür sind:

```
      ON ENDFILE(SYSIN)  GOTO XZ ;
      ON ENDFILE(SYSIN)  KLH=999 ;
MA:   ON ENDFILE(ZAHLUNG)  GOTO GES_PREIS ;
```

Aus den Beispielen ist ersichtlich, daß diese Bedingung durch das Schlüsselwort ENDFILE erkannt wird. Der Name der Datei, deren Dateiende das Auftreten dieser Bedingung signalisieren soll, ist hinter dem Schlüsselwort in Klammern aufzuführen.

Die ON-Anweisung braucht nur einmal ausgeführt zu werden. Durch sie wird die Maßnahme aufgesetzt, die beim Auftreten der Dateiendebedingung abläuft. Wird die Dateiendebedingung bei einer Datei festgelegt, für die keine entsprechende ON-Anweisung vorliegt und durchlaufen würde, so wird das Programm abgeschlossen.

Ein Statement beinhaltet nichts, d.h. es wird keine Operation ausgeführt. Die sogenannte Leeranweisung kann an jeder beliebigen Stelle eines in PL/1 geschriebenen Programmes erscheinen. Selbstverständlich kann es auch mit einer Marke versehen werden. Geschrieben wird es einfach als ein Semikolon, d.h. es erscheint als

```
;
```

in einem Programm.

Das Statement

```
END [prozedurname] ;
```

muß in jedem PL/1-Programm als letztes physisches Statement erscheinen. Es signalisiert dem Kompilierer, daß im Programm keine weiteren Statements folgen. Zur Ausführungszeit erhält das Betriebssystem die Ablaufsteuerung zurück, wenn ein END-Statement festgestellt wird.

Das Statement

```
RETURN ;
```

darf ebenfalls dazu verwendet werden, um die Programmablaufsteuerung zurück zum Betriebssystem zu führen. Es kann an beliebiger Stelle innerhalb eines Programmes stehen. Es ersetzt keineswegs die Funktion des END-Statements, dem Kompilierer das Ende des Programmes anzuzeigen, d.h. es muß noch einmal betont werden, daß jedes Programm mit einem END-Statement abgeschlossen werden muß. Allerdings besitzt das END-Statement zusätzlich die gleiche Funktion zum Verlassen eines Programmes wie die RETURN-Anweisung.

Die Ausführung des Statements

```
STOP ;
```

bewirkt den sofortigen Abbruch der Ausführung des Objektprogrammes. STOP wird als anormale Beendigung eines Programmes betrachtet, während END und RETURN zur normalen Beendigung eines Programmes eingesetzt werden.

In einem Programm können z.B. Steuerungsstatements in der Form und Anordnung codiert sein, wie es die Abb. 11.5. zeigt.

```
P1:     PROCEDURE OPTIONS(MAIN) ;
          ON ENDFILE(SYSIN)  GOTO ENDE ;
          ...
P7:       K = J ;   /* SCHLEIFENBEGINN */
          ...
          GOTO P7 ;
ENDE:     XVY7_IU = K*K + 4 ;
          ...
        END P1 ;
```

Abb. 11.5. Beispiele für Steuerungsstatements

11.10. Die listengesteuerte Eingabe/Ausgabe (EA)

Es gibt in PL/1 verschiedene Methoden, um Eingabe/Ausgabe-Operationen hervorzurufen. Die listengesteuerte reihenweise Eingabe/Ausgabe betrachtet die Eingabedaten als eine Reihe von Zeichen, wobei die zu jeder Eingabevariablen gehörende Folge von Zeichen von der nächstfolgenden durch ein Komma oder ein Leerzeichen getrennt ist; weitere Leerzeichen können außerdem vorhanden sein. Bei der Ausgabe werden die Werte ebenfalls als Reihe von Zeichen geschrieben, wobei die zu einer Variablen gehörenden Zeichen durch Leerzeichen von den Zeichen der nächstfolgenden Variablen getrennt sind.

Bei der Eingabe werden die Zeichen aus einer Quellen - oder Ausgangsdatei extrahiert, bei der Ausgabe in eine Zieldatei hineingestellt. Eine Datei wird mittels einer Vereinbarung festgelegt. Das DECLARE-Statement muß zu diesem Zweck wie folgt lauten:

DECLARE dateiname [FILE] [INPUT / OUTPUT] [PRINT] [INTERNAL / EXTERNAL]
[ENVIRONMENT(angaben)] ;

Die Aufführung eines der Attribute INPUT, OUTPUT, PRINT oder ENVIRONMENT schließt die Angabe von FILE mit ein, so daß dieses Attribut nicht unbedingt vorhanden sein muß.

Das Attribut INPUT besagt, daß es sich um eine Eingabedatei handelt, die Attribute OUTPUT bzw. PRINT, daß es sich um eine Ausgabedatei handelt. PRINT besagt zusätzlich, daß die Zieldatei einem Zeilendrucker zugeordnet ist, d.h. zu Beginn jeder Ausgabezeile erscheint ein ASA-Steuerzeichen [1]. Es genügt das Attribut PRINT bei Druckdateien vollauf. Auch INPUT kann wegfallen, wenn dateiname irgendwo im Pro-

1) Neuerdings auch als USASI- oder ANS-Steuerzeichen bezeichnet.

gramm in einem GET-Statement erscheint. Desgleichen ist die Angabe von OUTPUT überflüssig, wenn dateiname irgendwo im Programm in einer PUT-Anweisung steht.

Die Angaben beim ENVIRONMENT-Attribut schließen ein:

```
satzart
RECSIZE(satzlänge)
BLKSIZE(blocklänge)
BUFFERS(n)
LEAVE
REREAD
UNLOAD
```

Im Betriebssystem DOS kommen hinzu:

```
NOTAPEMK
NOLABEL
STACKER(n)
RCE
MEDIUM(SYSxxx ,gerätetyp])
```

Die meisten dieser Angaben sind optional, speziell im Betriebssystem OS. Sie können jederzeit durch äquivalente Angaben mittels der Jobsteuersprache ersetzt werden. Die Satzarten können in der Form F, FB, V, VB oder U spezifiziert werden. Die natürliche Zahl n bei BUFFERS legt die Anzahl der EA-Gebiete fest, die für die Datei bereitgestellt werden. Beim Betriebssystem DOS ist n auf 1 oder 2 zu beschränken, das Betriebssystem OS läßt für n Werte zwischen 1 und 25 zu. Die Angabe LEAVE besagt, daß nach dem Ende der Verarbeitung das die Datei enthaltende Magnetband auf das Dateiende eingestellt wird. REREAD hingegen, das nur im Betriebssystem OS angegeben werden kann, hat zur Folge, daß die Datei nach dem Ende der Verarbeitung wieder auf ihren Anfangspunkt eingestellt wird. Die Angabe von UNLOAD darf nur im Betriebssystem DOS erfolgen; sie bedeutet, daß das die Datei enthaltende Magnetband zurückgespult und entladen wird. Ist für eine Datei keine dieser drei zuletzt genannten Angaben gemacht worden, so wird das Magnetband so positioniert, wie es der DISP-Parameter des DD-Statements vorsieht (Betriebssystem OS) bzw. das Magnetband wird bis zum Ladepunkt zurückgespult (Betriebssystem DOS). Die Angabe NOLABEL wird so interpretiert, daß das entsprechende Magnetband keine Kennsätze (Etiketten) aufweist. Standardmäßig werden Kennsätze erwartet. Die Angabe NOTAPEMK darf sich nur auf Magnetbänder ohne Kennsätze beziehen; sie besagt, daß das Magnetband am Anfang keine Bandmarke besitzt. Durch die Angabe STACKER wird für Lochkartenstanzgeräte das Ablagefach spezifiziert, wobei n gleich 1 oder gleich 2 ist. Mit RCE ist das selektive Lesen beim Lochkartenleser IBM 3505 gemeint.

Die Angabe MEDIUM muß im Betriebssystem DOS unbedingt erscheinen. Durch die zweite Teilangabe wird der Typ des EA-Gerätes festgehalten; erlaubt sind hier z. B. 2540, 2501, 2314, 3330, 3420 usw. Diese Teilangabe kann entfallen, wenn bei SYSIPT oder SYSRDR die Blocklänge 80 angegeben ist bzw. bei SYSLST eine solche von 121 (einschließlich des Steuerzeichens). Auch bei SYSPCH kann die Angabe des Gerätetyps weggelassen werden, vorausgesetzt, daß die Blocklänge gleich 81 (einschließlich des Steuerzeichens) ist. Fehlt die 2. Teilangabe bei einer anderen logischen Einheit SYSxxx, so erzeugt der Kompilierer Befehle und Instruktionen, die für jede gültige Zuordnung von SYSxxx gelten.

Beim Betriebssystem OS ist es möglich, das DECLARE-Statement für einen Dateinamen gänzlich wegzulassen und alle gewünschten Dateibeschreibungen ins DD-Statement der Jobsteuersprache zu delegieren. Beim Betriebssystem DOS ist die DECLARE-Vereinbarung unbedingt erforderlich, um die Angabe MEDIUM im ENVIRONMENT-Attribut machen zu können.

Das grundlegende Eingabestatement lautet

GET FILE(dateiname) LIST(v_1,v_2,···,v_n) ;

Jedes v_i mit i=1,2,···,n bezeichnet eine Variable, für welche ein Wert eingelesen werden soll. Vom Eingabedatenträger wird Zeichen auf Zeichen gelesen. Zeichenketten und Bitketten innerhalb des Eingabestroms müssen dabei in Anführungszeichen (Hochkommas) eingeschlossen sein. Jeder Wert innerhalb des Eingabestroms muß vom nächstfolgenden durch ein Komma oder mindestens ein Leerzeichen getrennt sein. Nach dem Einlesen wird jeder Wert sofort in die Form umgewandelt, die der Erklärung der Variablen v_j, der er zugewiesen werden soll, gerecht wird. Ein Beispiel möge das illustrieren:

```
DECLARE  A DECIMAL FIXED(5,2),
         B DECIMAL FIXED(3,2),
         C DECIMAL FIXED(5,1),
         D CHARACTER (3) ;
GET FILE(LIEBE) LIST(A,B,D,C) ;
```

Durch diesen Programmabschnitt mögen Lochkarten gelesen werden, die wie folgt abgelocht sind:

```
2.3    3.45      'STT'   6.7
89  -4.22   'AABBCC'   1.Ø7    5,2
'U'   -3
```

Wird die GET-Anweisung zum ersten Mal ausgeführt, so erfolgen die Zuweisungen

```
ØØ2.3Ø ——→ A
  3.45 ——→ B
ØØØ6.7 ——→ C
   STT ——→ D
```

Stößt die Programmablaufsteuerung zum zweiten Mal auf diese Leseoperation, so erfolgen die Zuweisungen

```
Ø89.ØØ ——→ A
 -4.22 ——→ B
ØØØ1.Ø ——→ C
   AAB ——→ D
```

Interessant bei dieser zweiten Zuweisungsserie sind die Datenumwandlungen, der die Eingabewerte für C und D unterzogen werden. Bevor sie den genannten Variablen als Werte zugewiesen werden, unterliegen sie einer Abschneidung.

Eine dritte Ausführung der GET-Anweisung bringt folgende Wertzuweisungen:

```
 ØØ5.ØØ ——→ A
   2.ØØ ——→ B
-ØØØ3.Ø ——→ C
    Ubb ——→ D
```

Hierbei bedeutet b das Leerzeichen.

Bei der listengesteuerten EA werden Lochkartenbegrenzungen negiert. Man kann also den Wert, der für eine Variable bestimmt ist, am Ende einer Karte beginnen und über deren Begrenzung hinaus auf den ersten Spalten der nachfolgenden Lochkarte fortsetzen. Das rührt daher, weil bei der listengesteuerten EA die Eingabedaten als kontiniuerliche Folge von Zeichen angesehen werden, wobei physische Grenzen, wie Lochkartenanfang und -ende einfach unbeachtet bleiben.

Zur listengesteuerten Ausgabe dient das Statement

```
PUT FILE(dateiname) LIST(v1,v2,···,vn) ;
```

Zeilen- oder Kartenbegrenzungen werden hierbei ignoriert; alle Ausgabedaten belegen gerade den Platz auf dem Datenträger, den sie erfordern. Die einzelnen Ausgabewerte sind durch die listengesteuerte Eingabe/Ausgabe hauptsächlich dann angebracht, wenn der Programmierer sich nicht mit dem Format der Daten befassen muß. Das ist oft bei mathematischen und wissenschaftlichen Berechnungen der Fall, jedoch beinahe niemals dagegen bei kaufmännischen Problemen.

Die Option FILE(dateiname) kann sowohl bei den GET- als auch bei den PUT-Anweisungen weggelassen werden. Geschieht das, so beziehen sich die EA-Anweisungen auf zwei Standarddateien. Die Standardeingabedatei heißt SYSIN, die Standardausgabedatei, die zugleich die Eigenschaften einer Druckdatei hat, SYSPRINT. Beide Dateien brauchen nicht erklärt zu werden. Wenn SYSIN und SYSPRINT nicht anderweitig in einer DECLARE-Vereinbarung erscheinen, so gelten für sie die nachfolgenden impliziten Festlegungen, die wir der leichteren Lesbarkeit wegen in Form von DECLARE-Statements niederschreiben wollen.

```
DECLARE  SYSIN FILE INPUT
         ENVIRONMENT (F RECSIZE(8Ø) BLKSIZE(8Ø)
                      BUFFERS(2) MEDIUM(SYSIPT)) ;
DECLARE  SYSPRINT FILE OUTPUT PRINT
         ENVIRONMENT (F RECSIZE(121) BLKSIZE(121)
                      BUFFERS(2) MEDIUM(SYSLST)) ;
```

Diese Erklärungen sind auf das Betriebssystem DOS zugeschnitten, im Betriebssystem OS kann selbstverständlich die Angabe MEDIUM fehlen.

Wir haben also gesehen, was passiert, wenn die Option FILE(dateiname) in den EA-Statements weggelassen wird. Man kann aber in GET-Anweisungen auch FILE(SYSIN) und in PUT-Anweisungen FILE(SYSPRINT) schreiben, um den gleichen Effekt zu erzielen. Bei der ON-Anweisung muß sogar der Standarddateiname in der ENDFILE-Bedingung explizit erscheinen (siehe hierzu Abb. 11.5. im Abschnitt 11.9.).

In die Datenlisten der PUT-Anweisungen dürfen neben Variablen auch Konstanten aufgenommen werden. Somit ist das Statement

```
PUT LIST ('X =',X) ;
```

durchaus korrekt codiert und führt zum Drucken der Zeichenfolge

```
X = 3.45
```

wenn X zufällig den Wert 3.45 besitzt. Auf diese Weise können z.B. einfach und bequem Überschriften und Zwischentexte gedruckt werden.

Die GET-Anweisungen können zusätzlich mit der Option

```
COPY [(dateiname)]
```

ergänzt werden, wobei ihre Anordnung innerhalb der GET-Anweisung an beliebiger Stelle erfolgen kann. Diese Option bewirkt, daß die eingelesenen Daten anschließend über die Datei mit dem Dateinamen dateiname gedruckt werden. Fehlt die Angabe des Dateinamens, so erfolgt das Drucken über die Standarddruckdatei SYSPRINT. Die Druckausgabe selbst ist eine getreuliche Kopie der Eingabedaten. Besonders nützlich kann dieses Vorgehen für die Fehlersuche in Programmen sein.

11.11. Ein Beispielprogramm

Angenommen, wir haben mehrere Beobachtungen vorgenommen, deren Meßwerte sich als zwei- oder dreistellige ganze Zahlen ausdrücken lassen. Wir wollen nunmehr diese Werte einlesen und folgende Ergebnisse ausdrucken lassen:

a) die Zahl aller Beobachtungen,
b) die Summe aller Meßwerte der Beobachtungen und
c) die Summe der Quadrate aller Meßwerte. Diese Informationen werden zur Berechnung der Standardabweichung benötigt. Ein Programm, das dieses Problem löst, könnte den Aufbau besitzen, wie er in Abb. 11.6. dargestellt ist.

```
/* PROGRAMMBEISPIEL 11.11. --- BETRIEBSSYSTEM DOS */
   S:  PROCEDURE OPTIONS(MAIN) ;
         DECLARE  EDAT FILE INPUT
                       ENVIRONMENT (F RECSIZE(8Ø) BKLSIZE(8Ø)
                                    MEDIUM(SYSIPT)) ;
         DECLARE  ZAEHL FIXED(3,Ø),
                  SUMME FIXED(6,Ø),
                  X     FIXED(3,Ø),
                  WUSUM FIXED(9,Ø) ;
         ON ENDFILE(EDAT)  GOTO E ;
         ZAEHL = Ø ;   SUMME = Ø ;   WUSUM = Ø ;
 SCHL:   GET FILE(EDAT) LIST(X) ;
         ZAEHL = ZAEHL + 1 ;
         SUMME = SUMME + X ;
         WUSUM = WUSUM + X*X ;
         GOTO SCHL ;
     E:  PUT LIST (ZAEHL,SUMME,WUSUM) ;
       END ;
```

Abb. 11.6. Programmbeispiel

Beim Programmbeispiel von Abb. 11.6. wird durch das GET-Statement jeweils ein Wert von der Eingabedatei EDAT gelesen und der Variablen X zugewiesen; die Ausgabe erfolgt über die Standarddruckdatei SYSPRINT. Im Betriebssystem OS kann die Angabe MEDIUM im ENVIRONMENT-Attribut weggelassen werden.

Die auf die Bedingung ENDFILE(dateiname) bezugnehmende ON-Anweisung wird nur einmal ausgeführt. Sie setzt eine automatische Verzweigung nach F auf, wenn die Dateiendebedingung bei der Eingabedatei EDAT festgestellt wird. Bei diesem Programm gibt es sonst keine anderer Möglichkeit, aus der Schleife herauszukommen als das Auftreten der Dateiendebedingung.

11.12. Vergleiche

Vergleiche werden in der Form

operand$_1$ operator operand$_2$

geschrieben.

Die Operanden können arithmetische Daten, Zeichenkettendaten oder Bitkettendaten sein. Acht Operatoren sind erlaubt:

```
>     <     >=     =
¬>    ¬<    <=     ¬=
```

Ist ein Operand vom komplexen Typ, so sind nur die Operatoren = und ¬= möglich. Unterscheidet sich der Typ der beiden Operanden, so werden Bitketten in Zeichenketten und Zeichenketten in arithmetische Daten umgewandelt. Unterschiedliche Attribute bei zwei arithmetischen Operanden werden vor Durchführung des Vergleichs auf einheitliche Attribute ausgerichtet; hierbei erfolgt ein Übergang in der Richtung REAL nach COMPLEX, FIXED nach FLOAT und DECIMAL nach BINARY. Zeichenkettenvergleiche beruhen auf der Sortierfolge der gültigen Verschlüsselungen bei der IBM /370.

Der Vergleich arithmetischer Daten bzw. von Zeichenketten, die in arithmetische Daten umgewandelt werden, erfolgt auf algebraischer Basis. Der Vergleich von Zeichenketten erfolgt auf logischer Grundlage, die kürzere Kette wird hierbei nach rechts mit Leerzeichen aufgefüllt. Beim Vergleich von Bitketten wird die kürzere Kette ebenfalls nach rechts erweitert, jedoch mit Nullen.

Als Resultat eines Vergleiches ergibt sich eine Bitkette der Länge 1, die den Wert '1'B besitzt, wenn der Vergleich *wahr* ist und die den Wert '0'B besitzt, wenn der Vergleich *unwahr* bzw. *falsch* ist.

Die Operanden dürfen Ausdrücke, einfache Variablen oder Konstanten sein. Angenommen, es liegt folgender Programmabschnitt vor:

```
DECLARE   X FIXED(3,Ø),
          Y FLOAT(6),
          Z COMPLEX(6),
          H CHARACTER(5),
          B BIT(4) ;
X = -15 ;
Y = 7.8 ;
Z = 7.8 ;
H = 'CRTR5' ;
B = '11Ø1'B ;
```

Es erweisen sich dann z.B. folgende Vergleiche als *wahr:*

```
Y > X
Z = Y
H > 'ABCDE'
B ¬= '1111'B
```

Folgende Vergleiche erweisen sich z.B. als *falsch:*

```
X = Y
B = '1111'B
H ¬> 'A'
H > B
```

Die nachfolgenden vier Vergleiche sind ungültig und würden das Objektprogramm veranlassen, eine Fehlerbedingung zu signalisieren.

```
Z > Y
Z ¬> X
H > Y
H ¬> X
```

Bei den beiden letzten Vergleichen müßte z.B. die Zeichenkette 'CRTR5' in ein arithmetisches Datenelement umgewandelt werden, da arithmetische Daten, hier Y bzw. X,

in Vergleichen Vorrang vor den anderen Problemdatentypen haben. Da das unmöglich ist, muß ein solcher Vergleich entfallen. Wäre jedoch H der Wert '10110' zugewiesen, dann könnte diese Zeichenkette ohne weiteres in einen arithmetischen Wert umgewandelt werden. Unter dieser Voraussetzung würden dann die Vergleiche

```
H > Y
H ¬= X
H = B
```

usw. auch zu einer gültigen Aussage führen.

Soll eine Zeichenkette in ein arithmetisches Datenelement umgewandelt werden, so muß die Zeichenkette einen arithmetischen Wert in der gleichen Form enthalten, wie er in der Eingabe bei listengesteuerten Eingabe/Ausgabe-Operationen auftreten darf. Es sei jedoch darauf hingewiesen, daß derartige Umwandlungen zeitvergeudend sind.

Bei der Umwandlung von Bitketten in Zeichenketten wird jede Dualziffer 0 in das Zeichen 0 und jede Dualziffer 1 in das Zeichen 1 umgewandelt.

Das Ergebnis eines Vergleiches kann auf zweierlei Weise verwendet werden. Es kann durch eine reguläre Ergibtanweisung einer Variablen zugewiesen werden, z.B. wird durch

```
B = Y=Z ;
```

die Variable B auf den Wert '1ØØØ'B gesetzt, da der Vergleich Y=Z wahr ist und B eine Länge von vier Bitstellen aufweist. Ebenso wird durch

```
B = Y>X ;
```

die Variable B auf den Wert '1ØØØ'B gesetzt.

Das Vorhandensein eines Vergleichsoperators in einer Ergibtanweisung rechts von einem Ergibtzeichen bewirkt, daß der Kompilierer Instruktionen generiert, die den Vergleich realisieren.

In gleicher Weise setzt die Ergibtanweisung

```
B = (2*X+3Ø)=Ø ;
```

die äquivalent der Ergibtanweisung

```
B = 2*X+3Ø=Ø ;
```

ist, die Variable B auf den Wert '1ØØØ'B. Die letztere Schreibweise kann deshalb praktiziert werden, weil bei Fehlen von Klammern zur Festlegung der Reihenfolge von Operationen die arithmetischen Operationen zuerst ausgeführt werden. Sollten jedoch hinsichtlich der Reihenfolge von Operationen Zweifel bestehen, so ist es ratsam, immer Klammern zu benutzen.

Ein Vergleich wird sehr oft in einer IF-Anweisung gebraucht werden. Diese besitzt dann die Form

```
IF  bitkette  THEN s1 ;
              ELSE s2 ;
```

Unter bitkette ist eine Bitkette zu verstehen; gewöhnlich wird bitkette das einstellige Resultat eines Vergleiches sein. Ist der Inhalt von bitkette nicht gleich 'Ø'B, so wird das Statement s_1 ausgeführt, anderenfalls wird es übergangen und stattdessen wird das Statement s_2 ablaufen. Es ist zulässig, die ELSE-Klausel wegzulassen. Weiterhin ist zu beachten, daß vor dem Schlüsselwort THEN kein Semikolon stehen darf. Beispiele für typische IF-Anweisungen sind:

```
a)  IF  X>Y  THEN  B = '11ØØ'B ;
             ELSE  B = 'ØØØØ'B ;

b)  IF  B='111Ø'B  THEN ;
                   ELSE  H = 'ABCD7' ;

c)  IF  Y=Z  THEN  B='1111'B ;

d)  IF  X>Y  THEN    IF  Y=Z  THEN  B = 'ØØØ1'B ;
                                ELSE  B = 'Ø1ØØ'B ;
             ELSE    B = 'ØØ1Ø'B ;

e)  IF  X>Y  THEN  H = 'HIGH' ;
             ELSE  H = 'LOW' ;

f)  IF  X>Ø  THEN Y=+1 ;

g)  IF  B  THEN B='Ø1'B ;
           ELSE B='ØØ'B ;

h)  IF  Y=3  THEN ;
             ELSE Y=4 ;
```

Bei den Beispielen b) und h) wird das Statement, das auf das Schlüsselwort THEN folgt, als Leeranweisung bezeichnet, weil es keine Aktion irgendwelcher Art hervorruft. Es besteht also nur aus einem Semikolon. In den Beispielen c) und f) fehlt die ELSE-Klausel. Das Weglassen der ELSE-Klausel ist gleichbedeutend mit der Niederschrift von

```
ELSE ;
```

Die Niederschrift einer Leeranweisung kann man bei den Beispielen b) und h) durch Auswahl eines anderen Vergleichsoperators elegant umgehen. So umgestaltet, zeigen sich die beiden Statements in folgendem Gewand:

```
b)  IF  B¬='111Ø'B  THEN H='ABCD7' ;

h)  IF  Y¬=3  THEN Y=4 ;
```

Sollte jemand Spaß daran haben, so darf er selbstverständlich in beiden Fällen noch

```
ELSE ;
```

niederschreiben.

Es ist gestattet, in die THEN- oder ELSE-Klauseln von IF-Statements selbst wieder IF-Anweisungen aufzunehmen. Beispiel d) demonstriert das in eindrucksvoller Weise. Jede ELSE-Klausel wird hierbei mit der unmittelbar vorangehenden THEN-Klausel gepaart.

Erinnern wir uns: Wir hatten zunächst gesagt, daß nur ein einfaches Statement in den THEN- bzw. ELSE-Klauseln der IF-Anweisungen stehen darf. Natürlich darf auch ein GOTO-Statement dazu herangezogen werden; z. B. ist das bei

```
IF   X=5  THEN GOTO M5 ;
```

der Fall.

Aufgabengruppe 27

1) Das Programmbeispiel im Abschnitt 11.1. ist so abzuändern, daß auch Mittelwert und Varian berechnet und gedruckt werden!

2) Drucke eine Tabelle der x- und y-Werte für folgende Funktion!

$$y = \begin{cases} 2x^3 + 3 & \text{für } x < 0 \\ 3 - 4x & \text{für } 0 \leqq x \leqq 2 \\ 3 - 2x^2 & \text{für } x > 2 \end{cases}$$

Es ist mit x=-1 zu beginnen. Die Rechnungen sind fortzusetzen bis zu x=3 unter Einhaltung einer Schrittweite von 0,01 für die x-Werte.

3) Die Aufgabe 2) ist so zu modifizieren, daß zum nächsten Vielfachen von 0,05 für die x-Werte gesprungen wird, wenn die Differenz zweier aufeinanderfolgender y-Werte kleiner als 0,10 ist!

4) Es liegt ein Stapel gelochter Karten vor. Auf den Spalten 2 bis 10 sind x-Werte, auf den Spalten 12 bis 20 y-Werte abgelocht. Der Rest der Lochkarten bleibt leer. Schreibe ein Programm zum Lesen dieser Karten. Anschließend sind die Werte von a, b und r zu drucken. Hierbei ist y=a+bx die Regressionsgerade und r ist der Korrealtionskoeffizient (siehe hierzu Abschnitt 2.8.).

5) Es sind quadratische Gleichungen in der Form $ax^2 + bx + c = 0$ zu lösen. Die Koeffizienten a, b und c einer Gleichung sind hintereinander auf einer Karte abgelocht. Es soll versucht werden, die Gleichung dadurch zu lösen, daß für x der Reihe nach die Werte 0.00 , 0.01 , 0.02 usw. eingesetzt werden. Für jedes x sollen die zugehörigen Funktionswerte $f(x) = ax^2 + bx + c$ bestimmt werden. Werden dabei zwei aufeinanderfolgende Funktionswerte mit entgegengesetztem Vorzeichen gefunden, so ist dadurch eine Wurzel (Lösung) der Gleichung lokalisiert. Die nächste Berechnung ist dann mit einem x-Wert vorzunehmen, der zwischen den zuvor ermittelten zwei x-Werten liegt. Der Funktionswert des neuen x-Wertes ist dann mit demjenigen der beiden vorhergehenden x-Werte zu paaren, der das unterschiedliche Vorzeichen zu dem jetzt gefundenen Funktionswert besitzt. Zwischen den beiden zugehörigen x-Werten ist ein neuer x-Wert zu wählen, zu dem wiederum der Funktionswert bestimmt wird. Der Prozeß ist solange fortzusetzen, bis die Wurzel auf eine Differenz von 0.001 genau ermittelt ist. Es sind anschließend die Werte der Koeffizienten und die gefundene Wurzel der Gleichung zu drucken. Die Untersuchung ist auf die x-Werte zwischen 0 und 10 zu beschränken. Wird zwischen x=0 und x=10 keine Wurzel gefunden, so sollen die Koeffizienten sowie die Meldung

```
KEINE LOESUNG ZWISCHEN X=0 UND X=10
```

gedruckt werden.

11.13. Die datengesteuerte Eingabe/Ausgabe (EA)

Die Anweisungen

```
GET [FILE(dateiname)] DATA [COPY [(dateiname)]] ;
GET [FILE(dateiname)] DATA(v1,v2,···,vn) [COPY [(dateiname)]] ;
PUT [FILE(dateiname)] DATA(v1,v2,···,vn) ;
PUT [FILE(dateiname)] DATA ;
```

können zum Lesen bzw. Schreiben von Daten verwendet werden. Wird die Option FILE(dateiname) weggelassen, so wird der Dateiname SYSIN bei den GET-Statements und der Dateiname SYSPRINT bei den PUT-Statements angenommen. In der ersten Form des GET-Statements, bei der keine Variablen spezifiziert sind, wird die Datei so gelesen, als ob sie aus einer Folge von Zeichen bestehen würde. Die Eingabedaten erscheinen, um ein Beispiel zu geben, in folgender Form:

```
N=27, R=8.1, A=2.4   B=1.7E-3;
```

Die Variablen, für die neue Werte eingelesen werden sollen, erscheinen also mit ihrem Namen im Strom der Eingabedaten. Dem Namen folgt ein Ergibtzeichen, dem sich wiederum der neue Wert anschließt. Die Variablen werden durch ein Komma oder ein Leerzeichen voneinander getrennt. Die zur Eingabe durch ein einziges Statement vorgesehene Menge von Daten muß durch ein Semikolon abgeschlossen werden.

Wird im GET-Statement eine Liste von Variablen geschrieben, dann dürfen neue Werte nur solchen Variablen zugewiesen werden, die in der Liste aufgeführt sind. Wird der Versuch unternommen, einen neuen Wert für eine Variable einzulesen, deren Name nicht in der Liste steht, so tritt eine Fehlerbedingung auf. Andererseits ist es natürlich nicht notwendig, neue Werte für jede Variable der Liste einzulesen. So könnte die Eingabedatenreihe, die mit

```
GET DATA(A,B,C,D,E) ;
```

eingelesen werden soll, etwa wie folgt aussehen:

```
A=78 E=13.528,D=45;
```

Daraus würde überhaupt kein Problem entstehen. Es ist außerdem zu bemerken, daß die Variablen auf dem Datenträger in einer anderen Reihenfolge erscheinen dürfen als sie in der Liste aufgeführt sind; auch das ist ohne weiteres erlaubt.

Das PUT-Statement verursacht die Ausgabe aller in der Datenliste stehenden Variablen und der Werte, die sie im Augenblick der Ausgabe angenommen haben. Die einzelnen Ausgabelemente erscheinen wie bei der Eingabe in Form einer Ergibtanweisung und werden durch Leerzeichen voneinander getrennt. Dem letzten Datenelement folgt ein Semikolon. Fehlt bei der PUT-Anweisung die Datenliste, so werden alle im Programm vorkommenden Variablen mit ihren Werten ausgegeben.

Wie bei der listengesteuerten Eingabe/Ausgabe ist der Eingabe- bzw. Ausgabedatenträger als ununterbrochen fortgesetzt zu denken; Zeilen- oder Kartenbegrenzungen bleiben also unberücksichtigt. Somit kann sich eine Menge von Eingabedaten über mehrere Karten hinweg fortsetzen ebenso wie eine einzige Karte mehrere Mengen von Eingabedaten enthalten kann.

Wird die datengesteuerte EA verwendet, so benötigt das Objektprogramm zu ihrer Realisierung einen beträchtlich großen Hauptspeicheranteil. Dessen Umfang kann um einen gewissen Betrag reduziert werden, wenn alle GET- bzw. PUT-Statements mit Datenlisten versehen sind.

Die COPY-Option darf ebenfalls jedem GET-Statement zugefügt werden. Sie wird genau so gehandhabt wie bei der listengesteuerten EA (siehe Abschnitt 11.10.).

11.14. Die Anweisung DISPLAY

Die Anweisung

```
DISPLAY(ausdruck) ;
```

bewirkt, daß der in den Klammern stehende Ausdruck in eine Zeichenkette umgewandelt und danach über die Konsole ausgegeben wird. Die Zeichenkette darf beim System IBM /370 eine Länge von 72 Zeichen nicht überschreiten. Die Verarbeitung fährt während der Nachrichtenausgabe fort. Weil die Konsole ein sehr langsames Eingabe/Ausgabe-Gerät ist, sollte die DISPLAY-Anweisung nur für Nachrichten reserviert werden, die der Maschinenbedienung zugänglich gemacht werden sollen. Arithmetische Daten werden in Zeichenketten umgewandelt. Mehrere Datenelemente können zu einer

einzigen Zeichenkette verbunden werden, wenn man den Verkettungsoperator verwendet. Beispielsweise wird durch die Anweisung

```
DISPLAY ('NACHRICHT' || XZW) ;
```

auf der Konsole

```
NACHRICHT     784
```

herausgegeben werden, wenn XZW den Wert 784 besitzt.

Schreiben wir

```
DISPLAY (ausdruck) REPLY(variable) ;
```

nieder, so geht das Programm nach der Ausgabe der Nachricht solange in einen Wartezustand über, bis der Operateur die Antwort eingetastet hat. Der bei REPLY stehende Variablenname muß der einer Zeichenkettenvariablen sein. Ihr Längenattribut darf beim System IBM /370 keinen größeren Wert als 72 spezifizieren. Gibt der Operateur weniger Zeichen ein als das Längenattribut der Variablen ausweist, so wird der Rest der Zeichenkette in üblicher Weise durch Leerzeichen aufgefüllt. Die Antwort muß abgeschlossen sein, bevor die Verarbeitung wieder aufgenommen wird. Somit liegt hier eine besonders langsam ablaufende Operation vor.

11.15. Die formatgesteuerte Eingabe/Ausgabe (EA)

Wird die listen- oder die datengesteuerte Eingabe/Ausgabe in einem Programm verwendet, dann wählt der Kompilierer eine Ausgabeform, die den Attributen der Daten am besten angepaßt ist. Mitunter wünscht der Programmierer jedoch ein bestimmtes Format für die Ausgabe, damit er den praktischen Bedürfnissen besser gerecht werden kann. Die formatgesteuerte Eingabe/Ausgabe eignet sich dazu in vorzüglicher Weise. Durch eine die Datenliste kompensierende Formatliste ist Anordnung und Umfang der in den Dateien, d.h. auf den Datenträgern, stehenden Daten bestimmt. Für die Eingabe ist die entsprechende GET-Anweisung wie folgt niederzuschreiben:

```
GET [FILE(dateiname)] EDIT {(v1,v2,···,vn) (f1,f2,···,fm)}···
                    [COPY [(dateiname)]] ;
```

Fehlt die FILE-Option, so wird wie üblich FILE(SYSIN) unterstellt. Die COPY-Option hat die gleiche Bedeutung wie bei der listen- bzw. datengesteuerten EA. Durch die v_i mit $i=1,2,\cdots,n$ wird die Datenliste gebildet, die Formatelemente f_j mit $j=1,2,\cdots,m$ formen die Formatliste. Die Zahl der Formatelemente braucht mit der Zahl der Datenelemente in der Datenliste nicht übereinzustimmen.

Die formatgesteuerte Ausgabeanweisung besitzt folgende allgemeine Form:

```
PUT [FILE(dateiname)] {EDIT(a1,a2,···,an) (f1,f2,···,fm)} ··· ;
```

Fehlt die Option FILE(dateiname) , so wird die Ausgabe auf die Standarddruckdatei SYSPRINT gelenkt, d.h. die Option FILE(SYSPRINT) ergänzt.

Wie bei den beiden anderen Formen der reihenweisen EA, d.h. der listen- bzw. datengesteuerten, wird auch hier angenommen, daß die Daten auf dem externen Medium eine kontinuierliche Reihe von Zeichen bilden. Diese Zeichenreihe wird jedoch auf der

Basis der Formatelemente und nicht auf der Basis von Trennzeichen in Felder unterteilt. Datenbeschreibende Formatelemente müssen deshalb die Feldlänge w als unumgänglich notwendige Angabe einschließen. Je ein Formatelement f_i wird zur Beschreibung der Form eines Datenelementes v_i (bei Eingabedatenlisten) bzw. eines Ausdruckes a_i (bei Ausgabedatenlisten) herangezogen. Neben diesen sogenannten Datenformatelementen können in eine Formatliste zusätzlich noch Steuerungsformatelemente aufgenommen werden. Diese stehen in keiner Beziehung zu Angaben in der Datenliste. Sie dienen vielmehr dazu, bestimmte Steuerungsfunktionen, wie Überspringen von Feldern, Einstellen auf eine bestimmte Stelle usw. auszuüben.

Die zwischen dem Hauptspeicher und den Externspeichern anfallenden Datenumwandlungen sind allein durch das zugeordnete Formatelement bestimmt. Die Reihenfolge, in der die Eintragungen (Datenelemente bzw. Ausdrücke) in der Datenliste erscheinen, bestimmte auch die Reihenfolge des Lesens bzw. des Schreibens der Daten. Die Formatelemente werden dabei ebenfalls sequentiell zur Interpretation herangezogen.

Für arithmetische Daten sind die folgenden Formatelemente verfügbar:

a) F(w,d)
Das Eingabefeld besteht aus w Zeichen, die eine Dezimalzahl darstellen. Ist in diesem Feld kein Dezimalpunkt fixiert, so wird die Zahl als ganzzahlig angesehen. Jeder negativen Zahl muß ein Minuszeichen vorangehen. Leerzeichen dürfen der Zahl sowohl vorausgehen als auch nachfolgen. Sind sie vorhanden, so bleiben sie unbeachtet. Ein Eingabefeld, das nur Leerzeichen enthält, wird als Zahl Null angesehen.

Bei der Ausgabe wird in das w Stellen breite Feld eine ganze Dezimalzahl ohne Dezimalpunkt ausgegeben. Führende Nullen werden bei der Ausgabe unterdrückt. Die auszugebende Zahl wird rechtsbündig in das Feld hineingestellt. Negativen Zahlen geht ein Minuszeichen voraus; es erscheint unmittelbar vor der ersten von Null verschiedenen Ziffer der Zahl. Bei einer Null wird rechtsbündig die Ziffer 0 geschrieben, nach links werden bis zur vollen Feldbreite Leerzeichen ergänzt.

b) F(w,d)
Wenn das w Stellen breite Feld bei der Eingabe keinen Dezimalpunkt enthält, so wird angenommen, daß die d rechten Stellen dieses Feldes die nach dem Dezimalpunkt stehenden Ziffern der Zahl bedeuten. Wird jedoch im Eingabefeld ein Dezimalpunkt entdeckt, so überlagert seine Position die im Formatelement angegebene Spezifikation d. Im Ausgabefeld erscheint grundsätzlich ein Dezimalpunkt. Rechts von ihm erscheinen d Ziffern, die die Dezimalstellen des auszugebenden Wertes repräsentieren und vor dem Dezimalpunkt erscheinen weitere w-d-1 Ziffern bei negativen Zahlen, und zwar jeweils die Ziffern des ganzzahligen Anteils der Zahl. Bei negativen Zahlen steht unmittelbar vor der ersten Ziffer ein Minuszeichen. Führende Nullen werden unterdrückt.

c) F(w,d,s)
Bei der Eingabe wird der Wert, der vom externen Medium gelesen wird, mit 10^s multipliziert, bevor er der Umwandlung in die interne Form unterworfen wird. Bei der Ausgabe ist der Ausgabewert gleich dem 10^s-fachen des internen Wertes. Die Spezifikation s wird aus diesem Grund als Skalenfaktor bezeichnet.

Zur Erläuterung des Skalenfaktors wollen wir uns das Beispiel in Abb. 11.7. ansehen.

```
   .
   .
   .
   R = 7.812 ;
   S = -213Ø ;
   T = 45. ;
   U = Ø ;
   PUT EDIT (R,S,T,U) (F(6,3),F(7,3,-3),F(5),F(2)) ;
   .
   .
   .
   GET EDIT (R,S,T,U) (F(6,3),F(7,3,-3),F(5),F(2)) ;
   .
   .
   .
```

Abb. 11.7. Skalenfaktoren bei F-Formatelementen

Die durch das PUT-Statement in die Standarddruckdatei SYSPRINT veranlaßte Ausgabe besteht aus folgenden 20 Zeichen:

```
b7.812b-2.13Øbbb45bØ
```

Hierbei bedeutet b das Leerzeichen.

Lesen wir diese Zeichenfolge durch das o. a. GET-Statement wieder ein, so werden den Variablen folgende Werte zugewiesen:

```
R=7.812
S=Ø.ØØ213
T=45
U=Ø
```

Besser als bei der Variablen S kann man wohl den Einfluß des Skalenfaktors auf den Wert eines Datenelementes nicht demonstrieren.

Für externe Felder zur Aufnahme von Zahlen in Gleitpunktform stehen die folgenden Formatelemente zur Verfügung.

d) E(w, d)

Das externe Medium enthält eine Zahl in Gleitpunktdarstellung. Bei der Eingabe sind folgende Formen möglich:

```
+1.234E+Ø7          -1.234E-Ø7
b1.234EØ7           b-1234E-7
b1.234E7            b-1234-7
b1.234+7            -1.234-Ø7
```

Unter b ist auch hier das Leerzeichen zu verstehen.

Es ist nur erforderlich, daß wenigestens ein E oder ein + oder ein - vor dem Exponenten stehen muß. Enthält das Eingabefeld keinen Dezimalpunkt, so wird durch d die Anzahl der Stellen nach dem Dezimalpunkt spezifiziert. Bei der Ausgabe belegt der Exponent stets vier Stellen; er tritt nämlich in der Form E±xx auf. Außerdem enthält die Ziffernfolge der Zahl gerade d Dezimalstellen. Das Vorzeichen der Zahl, entweder - oder b (bei positiven Zahlen) erscheint unmittelbar vor der ersten Ziffer. Vor dem Dezimalpunkt wird eine Ziffer ausgegeben. Die Zahl selbst wird rechtsbündig in das Feld mit der Breite w gestellt. Es braucht kein Zwischenraum vor der Zahl für das Vorzeichen vorgesehen zu werden, wenn die Zahl positiv ist.

e) E(w,d,s)
Bei der Eingabe bleibt s unbeachtet. Der Dezimalpunkt wird d-1 Stellen vor der letzten, dem Exponenten vorausgehenden Ziffernstelle angenommen. Enthält das Eingabefeld jedoch einen Dezimalpunkt, so gilt er als vorrangig gegenüber der im Formatelement angegebenen Spezifikation. Bei der Ausgabe wird die Zahl mit s-d Stellen vor dem Dezimalpunkt ausgegeben, d Stellen folgen dem Dezimalpunkt. Der Exponent selbst belegt grundsätzlich vier Stellen in der Form E±xx. Ein Minuszeichen geht der ersten Ziffernstelle bei negativen Zahlen voraus. Wenn wir uns auf das beim Formatelement F(w,d,s) aufgeführte Beispiel beziehen, so würde durch das Statement

```
PUT EDIT (R,S,T,U) (E(1Ø,4),E(1Ø,2),E(1Ø,Ø,2),E(1Ø,2)) ;
```

folgende Ausgabezeichenfolge entstehen:

```
7.812ØE+ØØb-2.13E+Ø3bbb45.E+ØØbbØ.ØØE+ØØ
```

Hierbei bedeutet b das Leerzeichen.

Aus dem soeben Gesagten ergibt sich, daß dem Formatelement E(w,d) das Formatelement E(w,d,d+1) gleichkommt.

Gibt man für arithmetische Ausgabewerte die Feldlänge w größer an als es unbedingt notwendig ist, so wird dadurch sicher ihre Lesbarkeit erheblich verbessert, da dann vor den Ziffern Leerzeichen erzeugt werden. Wird dagegen die Anzahl der Dezimalstellen durch entsprechende Angabe von d reduziert, dann werden die Ausgabewerte gerundet; es werden also nicht einfach nur die Ziffern niedrigster Ordnung abgeschnitten.

Neben den bisher besprochenen Formatelementen dürfen noch die folgenden Formatelemente verwendet werden:

f) B(w)
Eine aus w Zeichen bestehende Folge wird vom externen Medium gelesen und in eine interne Bitkette umgewandelt. Bei der Ausgabe wird eine Bitkette in eine aus den Zeichen 0 und 1 bestehende Zeichenkette umgewandelt und linksbündig in ein w Stellen breites Feld gestellt. Ist die Bitkette kürzer, so daß das w Stellen breite Feld nicht gefüllt werden kann, so wird das Feld nach rechts durch Leerzeichen aufgefüllt.

g) B
Dieses Formatelement ist nur für die Ausgabe definiert. Die Feldbreite w wird hierbei so bestimmt, daß sie die in eine Zeichenkette umgewandelte Bitkette komplett aufnehmen kann.

Für Zeichenketten gelten die nachfolgenden Formatelemente:

h) A(w)
Bei der Eingabe wird eine Folge von w Zeichen in eine Zeichenkettenvariable gelesen. Ist die Zeichenkette mit einer Länge erklärt, die größer als w ist, so werden die rechten Stellen der Variablen durch Leerzeichen aufgefüllt. Ist die erklärte Länge der Zeichenkettenvariablen jedoch kleiner als w, so wird die eingelesene Folge auf w Zeichen verkürzt, indem die rechten Stellen abgeschnitten werden. Bei der Ausgabe wird die interne Zeichenkette linksbündig in das Ausgabefeld gestellt, das bei Bedarf nach rechts mit Leerzeichen aufgefüllt wird. Ist w kleiner als die Länge der internen Kette, so wird diese rechts abgeschnitten, bevor sie ins Ausgabefeld gestellt wird. Die durch dieses Formatelement angesprochenen Variablen sollten möglichst Zeichenkettenvariablen sein.

i) A

Dieses Formatelement ist nur bei der Ausgabe erlaubt. Die Feldbreite w des Ausgabefeldes wird dabei gleich der Länge der internen Zeichenkette gesetzt. Beispielsweise resultiert der Programmabschnitt

```
DECLARE  CH CHARACTER(4), BI BIT(4) ;
   .
   .
   .
CH = '2BK=' ;
BI = 'ØØ11'B ;
PUT EDIT (CH,BI,'FEST') (A(6),B,A) ;
```

in der nachstehenden Folge von Ausgabezeichen:

2BK=ØØ11bbFEST

Wie üblich bedeutet b hier das Leerzeichen.

Für komplexe arithmetische Daten gibt es die folgenden Formatelemente:

j) C(r)
k) C(r_1,r_2)

Unter r bzw. r_1 und r_2 sind Formatelemente zu verstehen, mit denen die Darstellung reeller arithmetischer Werte auf dem Datenträger beschrieben werden kann, d.h. E- bzw. F-Formatelemente.

Wird in der Klammer nur eines der reellen Formatelemente mit den Kennbuchstaben F oder E angegeben, so gilt dieses für beide Teile der komplexen Zahl. Werden dagegen zwei Elemente spezifiziert, so gilt das erste für den Realteil und das zweite für den Imaginärteil der komplexen Zahlen. Grundsätzlich sind sowohl bei der Eingabe als auch bei der Ausgabe als reelle Formatelemente nur die E- und F-Elemente erlaubt.

Sind in einer Formatliste Formatelemente zu wiederholen, beispielsweise ···, E(11,4), E(11,4), ··· so kann eine abgekürzte Schreibweise benutzt werden, indem man einen Wiederholungsfaktor verwendet; dieser muß vor das zu wiederholende Formatelement in Klammern niedergeschrieben werden, hier also ···, (2)E(11,4), ···

Mitunter werden gleiche Formatlisten oder Teile derselben in mehr als einem GET- oder PUT-Statement benutzt. Man kann das wiederholte Schreiben derartiger Formatspezifikationen vermeiden, indem man ein sogenanntes R-Formatelement codiert.[1] Das drückt sich dadurch aus, daß man an Stelle der Formatliste bzw. eines Teiles derselben

R(marke)

codiert. Hierbei versteht man unter marke die Marke eines sogenannten FORMAT-Statements. Dieses Statement besitzt die Form:

marke:FORMAT(f_1,f_2,···,f_n);

Hierbei bedeuten die f_j beliebige der oben vorgestellten Formatelemente. Um diese Programmiertechnik zu erläutern, soll folgendes Beispiel betrachtet werden:

```
          .
          .
          .
          PUT EDIT (R,S,T) (R(FORM1),F(5)) ;
          GET EDIT (R,S,T) (R(FORM1),F(5,Ø)) ;
FORM1:    FORMAT (F(6,3),F(7,3,-3)) ;
          .
          .
          .
```

1) Aus dem angloamerikanischen Sprachraum von remote (entfernt).

Eingabe

Zeichenfolge in der Datei(externes Speichermedium)	Formatelement	Attribute der Variablen	Wert der Variablen im Hauptspeicher nach der Eingabe
ABCDE	A(1) A(2) A(3) A(5)	CHARACTER(3)	Abb ABb ABC ABC
b11b3	B(Ø) B(1) B(2) B(3) B(4) B(5)	BIT(4)	ØØØØ Umwandlungsfehler 1ØØØ 11ØØ 11ØØ Umwandlungsfehler
bb31.32	F(7) F(7,Ø) F(7,3) F(7,3,1)	DECIMAL FIXED(5,2)	Ø3 13 2+ Ø3 13 2+ Ø3 13 2+ ØØ 31 3+
bbbbb	F(7)	DECIMAL FIXED(5,2	ØØ ØØ Ø+
bb-478b	F(7) F(7,3) F(7,3,2)	DECIMAL FIXED(5,2)	47 8Ø Ø- ØØ Ø4 7- Ø4 78 Ø-
b1.234E-Ø7 b1234E-Ø7b bbb1234E-7 bb-87+8bbb bbbb-8.7E5 bbbbbbbbbb	E(1Ø,2)	DECIMAL FLOAT(6)	$.1234 \cdot 1Ø^{-6}$ $.1234 \cdot 1Ø^{-5}$ $.1234 \cdot 1Ø^{-5}$ $-.87 \cdot 1Ø^{8}$ $-.87 \cdot 1Ø^{6}$ Umwandlungsfehler

Ausgabe

Wert der Variablen im Hauptspeicher vor der Ausgabe	Attribute der Variablen	Formatelement	Zeichenfolge in der Datei (externes Speichermedium)
ABC	CHARACTER(3)	A A(2) A(3) A(4)	ABC AB ABC ABCb
11Ø1	BIT(4)	B B(3) B(4) B(5)	11Ø1 11Ø 11Ø1 11Ø1b
Ø3 13 2+	DECIMAL FIXED(5,2)	F(7) F(7,3) F(7,3,-1) F(7,3,2)	bbbbb31 b31.32Ø bb3.132 132.ØØØ
ØØ ØØ Ø+	DECIMAL FIXED(5,2)	F(7,3)	bbØ.ØØØ
Ø4 76 8+	DECIMAL FIXED(5,2)	F(7) F(7,Ø)	bbbb-48 bbb-48.
-.456789E8	DECIMAL FLOAT(6)	E(7,3) E(1Ø,2) E(1Ø,2,4)	-47.68Ø b-4.57E+Ø7 -45.68E+Ø6

Anmerkung: b bedeutet hier das Leerzeichen.

Abb. 11.8. Beispiel für die formatgesteuerte EA

Durch das R-Formatelement werden in das PUT- und in das GET-Statement zwei Formatelemente, nämlich die in der FORMAT-Vereinbarung mit der Marke FORM1 stehenden F(6,3) und F(7,3-3), hineingestellt, die dann die äußere Form der beiden Variablen R und S beschreiben. Es ist also gewissermaßen so, als ob die Formatliste ein „entfernt liegendes" Formatelement enthält. Innerhalb einer Formatliste dürfen beliebig viele R-Formatelemente aufgeführt werden.

Beim Schreiben aller Formatelemente dürfen an Stelle von w, d und s beliebige Ausdrücke eingesetzt werden, die zunächst berechnet und anschließend in eine ganze Zahl umgewandelt werden. Auf diese Weise ist es möglich, die Feldlänge (Feldbreite), die Position von Zeichen innerhalb der Ausgabezeichenfolge usw. noch während des Programmablaufes zu verändern.

Wir haben bis jetzt die Mehrzahl derjenigen Formatelemente kennengelernt, die Eintragungen in der Datenliste zugeordnet werden. Nunmehr wollen wir unser Augenmerk den Steuerungsformatelementen schenken, denen keine Eintragung in der Datenliste entspricht.

a) X(w)
Bei der Eingabe werden w Zeichen übergangen, bei der Ausgabe werden w Leerstellen erzeugt.

b) COLUMN(n)
Dieses Formatelement bewirkt, daß die Zeichenreihe auf dem externen Speichermedium, d.h. in der Datei, auf die n-te Stelle der augenblicklichen bzw., wenn diese schon überschritten ist, der nächsten Zeile (Lochkarte, Druckzeile etc.) eingestellt wird. Somit bedeutet z.B.

```
COLUMN(18),
```

daß ein unmittelbar folgendes Datenformatelement ein Feld beschreibt, das auf der Spalte 18 einer Eingabelochkarte, auf der Stelle 18 einer Druckzeile usw. beginnt.

Steuerungsformatelemente können durch Wiederholungsfaktoren genau so angesprochen werden wie Datenformatelemente. Ebenso dürfen sie in R-Formatelemente einbezogen werden. Weitere Steuerungsformatelemente werden wir im nächsten Abschnitt besprechen.

Wenn eine EA-Operation beginnt, so wird die Datenliste nach ihrer ersten Eintragung durchsucht. Danach wird das erste Formatelement in der Formatliste ermittelt. Ist dieses ein Steuerungsformatelement, so wird die Steuerungsfunktion ausgeführt und das Durchsuchen der Formatliste fortgesetzt. Nach Entdeckung eines Datenformatelementes wird die Umwandlung des Wertes besorgt, der zur Eintragung in der Datenliste gehört. Anschließend wird in der Datenliste nach der nächsten Eintragung gesucht. Dieses Pendeln zwischen Datenliste und Formatliste geschieht solange, bis die letzte Eintragung in der Datenliste abgearbeitet ist. Sollte zwischenzeitlich die Formatliste erschöpft sein, so wird die Formatliste erneut vom Anfang an durchsucht. Geht jedoch die Datenliste eher zu Ende als die Formatliste, so bleiben die restlichen Formatelemente unbeachtet. Was das bedeutet, möge das nachfolgende Beispiel zeigen:

```
PUT EDIT(R,S,T) ((3)F(7,4),X(8)) ;
```

Die Werte von R, S und T werden in ein sieben Stellen breites Feld (mit vier Dezimalstellen, einem Punkt und zwei Stellen vor dem Dezimalpunkt) ausgegeben. Zwischen R und S bzw. S und T befindet sich jeweils ein acht Stellen breiter Zwischenraum mit Leerzeichen. Der Wiederholungsfaktor 3 in der Formatliste gilt auch für X(8). Das letzte Mal (nach der Aufbereitung und Ausgabe des zu T gehörenden Wertes) wird je-

doch dieses Steuerungsformatelement nicht mehr beachtet, da die Datenliste mit T bereits ausgeschöpft ist.

In der Abb. 11.8. sind einige Beispiele für die Aufbereitung von Daten gemäß der ihnen zugeordneten Formatelemente aufgeführt.

11.16. Druckdateien

Wenn eine Ausgabedatei entweder direkt oder auf dem Umweg der Zwischenspeicherung auf Magnetband oder Magnetplatte gedruckt werden soll, so muß die Datei mit dem Attribut PRINT erklärt sein. Die Standardausgabedatei SYSPRINT hat von Haus aus dieses Attribut, ist also eine Druckdatei.

Bei Druckdateien kann das PUT-Statement durch eine der folgenden Optionen ergänzt werden:

```
PAGE
PAGE LINE(ausdruck)
SKIP
SKIP(ausdruck)
LINE(ausdruck)
```

Die Option SKIP (ausdruck) kann auch für andere Eingabe- oder Ausgabedateien niedergeschrieben werden; sie ist also nicht auf Druckdateien beschränkt.

Wird ein Ausdruck ausdruck niedergeschrieben, so wird er berechnet und danach in eine ganze Zahl n umgewandelt. Ist n kleiner als 1, so wird der Wert von n im allgemeinen durch 1 ersetzt. Eine Ausnahme bildet hierbei der Ausdruck in der SKIP-Option bei Druckdateien: hier wird die sich aus der Berechnung des Ausdruckes ergebende ganze Zahl n durch Ø ersetzt, falls sie kleiner als Null ist. PAGE bedeutet Vorschub auf die erste Druckzeile der nächsten Seite. Die Angabe der Option LINE bewirkt einen Formulartransport bis zur n-ten Zeile der betreffenden Seite; somit bedeutet z.B. LINE(5), daß die nächste Zeile, die bedruckt wird, die 5. Zeile der laufenden Seite ist. SKIP oder SKIP(1) haben zur Folge, daß der Druckvorgang auf der nächsten Zeile beginnt. SKIP(ausdruck) verursacht (n-1) Leerzeichen vor der nächsten zu bedruckenden Zeile. Überschreitet ein durch SKIP hervorgerufener Zeilentransport eine Seitenbegrenzung, so gleicht diese Option in ihrer Wirkung der Option PAGE. Die Formularbewegung findet stets vor dem nächsten Druckvorgang statt, gleichgültig an welcher Stelle die entsprechenden Optionen in der PUT-Anweisung aufgeführt sind. SKIP(Ø) verursacht, daß die soeben gedruckte Zeile noch einmal beim nächsten Druckvorgang bedruckt wird.

Bei der über eine Druckdatei erfolgenden formatgesteuerten Ausgabe dürfen die o.a. Optionen auch als Formatelemente in die Formatliste eingeschlossen werden. In diesem Fall werden die Formularbewegungen in dem Augenblick durchgeführt, in dem die Formatliste bis zu dem betreffenden Druckformatelement abgearbeitet ist. Dadurch ist es möglich, durch ein einziges PUT-Statement mehr als eine Zeile zu drucken. Beispielsweise ruft die FORMAT-Vereinbarung

```
m:  FORMAT (SKIP(2),F(6,3),F(8),SKIP(2),F(5)) ;
```

nacheinander folgende Vorgänge hervor:

a) Zweizeiliger Zeilentransport durch SKIP(2)
b) Ausgabe zweier Werte gemäß der Formatelemente F(6,3) und F(8)
c) Zweizeiliger Zeilentransport durch SKIP(2)
d) Ausgabe eines Wertes gemäß des Formatelementes F(5)

Es werden hier also zwei Zeilen ausgegeben.

Beispiele für die Optionen und Formatelemente folgen jetzt:

```
1)  PUT PAGE LIST('      X          Y') ;
2)  PUT SKIP LIST(X,Y) ;
3)  PUT LINE(56) LIST(SUMMEX,SUMMEY) ;
4)  PUT FILE(DAT) EDIT(X,Y) (F(6,2),F(1Ø,2)) ;
5)  PUT EDIT(X,Y,SUMMEX,SUMMEY)
          ((2)(F(6,2),F(1Ø,2),SKIP(3))) ;
```

Beim Beispiel 5) wird das in der Formatliste enthaltene Steuerungsformatelement SKIP(3) nur ein einziges Mal angezogen. Diese Aussage ergibt sich aus der Besprechung des Zusammenwirkens zwischen Datenliste und Formatliste im vorigen Abschnitt.
Das Beispiel 1) könnte zum Drucken von Überschriften dienen. Das zweite Beispiel bewirkt den Übergang auf eine neue Zeile und das Drucken der Werte von X und Y (bei passender Ausrichtung spaltengerecht unter der Überschriftszeile). Durch das 3. Statement werden auf die 56. Zeile einer Seite die Werte der Variablen SUMMEX und SUMMEY gedruckt. Alle Beispiele mit Ausnahme des vierten sind Anweisungen, die den Druckvorgang auf die Standarddruckdatei SYSPRINT beziehen.

Druckdateien müssen mit einer Blocklänge spezifiziert werden, die um eins größer als die Anzahl der Druckstellen pro Zeile ist. Nur dadurch ist Platz für das ASA-Steuerzeichen [1] gegeben. Das erste Zeichen eines Satzes wird also nie gedruckt; es steuert nur den Formulartransport vor dem eigentlichen Druckvorgang. Nur die folgenden Steuerzeichen sind zulässig:

1	Vorschub auf neue Seite
b	Einfacher Zeilentransport [2]
Ø	Doppelter Zeilentransport
-	Dreifacher Zeilentransport
+	Unterdrückung des Zeilentransportes

Die Steuerzeichen werden aus den Druckoptionen [3] bzw. aus den Formatelementen erzeugt.

In PUT-Anweisungen, die sich auf Druckdateien beziehen, dürfen die Datenspezifikationen LIST, DATA oder EDIT fehlen und allein Druckoptionen erscheinen. Z.B. dürfen wir ohne weiteres nur

```
PUT PAGE ;                    oder
PUT FILE(DAT) LINE(56) ;
```

schreiben, wodurch wir eine Formulareinstellung ohne nachfolgenden Druckvorgang erreichen.

Druckdateien können auch für Lochkartenstanzgeräte wie die IBM 3525 benutzt werden, um dort das Drucken zu steuern.

---

1) Auch als USASI- oder ANS-Steuerzeichen bezeichnet.
2) b bedeutet hier das Leerzeichen.
3) Eine Option entspricht einer Wahlfreiheit (Wahlmöglichkeit), d.h. es ist dem Benutzer freigestellt, das betreffende Sprachelement zu benutzen oder nicht zu benutzen.

Bei Dateien, die nicht als Druckdateien erklärt sind, bedeutet SKIP(ausdruck), daß (n-1) Sätze bei der Eingabe übergangen bzw. daß (n-1)nur aus Leerzeichen bestehende Sätze bei der Ausgabe erzeugt werden.
Bei allen Programmierungsarbeiten sollte nicht vorausgesetzt werden, daß zu Beginn des Programmablaufes das Papier auf den Anfang einer neuen Seite vorgeschoben ist.

11.17. Das Eröffnen und Abschließen von Dateien

Bevor das Betriebssystem eine Datenmenge verarbeiten kann, muß die betreffende Datei eröffnet werden. Nach Abschluß der Verarbeitung muß sie auch wieder abgeschlossen werden. Das Eröffnen und Abschließen kann implizit vorgesehen werden, d.h. der Programmierer braucht dazu keine gesonderte Anweisung niederzuschreiben. Das implizite Eröffnen geschieht dann während der Ausführung der ersten eine Datei ansprechenden GET- bzw. PUT-Anweisung. Das implizite Abschließen hingegen wird erst nach Abschluß der Verarbeitung, d.h. am Programmende vorgenommen.
Durch besondere Anweisungen kann der Programmierer auch explizit Dateien eröffnen und abschließen. Gezielt eröffnet werden Dateien durch das Statement:

```
OPEN FILE(dateiname) [optionen] [,FILE(dateiname) [optionen]]
```

Unter optionen können u.a. folgende Optionen aufgeführt werden:

```
INPUT | OUTPUT
LINESIZE(ausdruck)
PAGESIZE(ausdruck)
```

Die Optionen INPUT bzw. OUTPUT können die Festlegungen der entsprechenden Attribute in der DECLARE-Vereinbarung für die Datei ersetzen, d.h. die betreffenden Attribute werden der Datei erst beim Eröffnen zugeordnet. Der sich aus der Berechnung der bei LINESIZE bzw. PAGESIZE ergebende und in eine ganze Zahl n umgewandelte Wert hat folgende Bedeutung: Bei LINESIZE gebraucht, legt er die maximale Stellenzahl pro Zeile fest, bei PAGESIZE die maximale Zeilenzahl pro Seite. Beide Optionen dürfen nur Druckdateien spezifizieren. Bei Fehlen dieser Optionen wird standardmäßig angenommen, daß eine Druckseite aus 60 Zeilen und jede Zeile aus 120 Druckstellen besteht.

Eine OPEN-Anweisung, die sich auf eine bereits eröffnete Datei bezieht, wird ignoriert. Mit einer einzigen OPEN-Anweisung können beliebig viele Dateien eröffnet werden, wie das nachfolgende Beispiel zeigt:

```
OPEN FILE(MARI), FILE(ANNE) PAGESIZE(56),
     FILE(WUER) LINESIZE(132) PAGESIZE(17) ;
```

Eröffnete Dateien können explizit mittels der Anweisung

```
CLOSE FILE(dateiname) ENVIRONMENT(angabe)|
      [,FILE(dateiname) ENVIRONMENT(angabe)] ... ;
```

abgeschlossen werden.

Die einzigen Angaben, die in der ENVIRONMENT-Option getätigt werden dürfen, sind:

Betriebssystem OS	Betriebssystem DOS
LEAVE	LEAVE
REREAD	UNLOAD

Eine CLOSE-Anweisung, die sich auf eine nicht eröffnete Datei bezieht, gilt als nicht geschrieben. Fehlt die ENVIRONMENT-Option in der CLOSE-Anweisung, so tritt an

ihre Stelle das ENVIRONMENT-Attribut in der DECLARE-Vereinbarung, falls ein solches vorhanden ist.

Da mit Hilfe der OPEN-Anweisung Seiten- und Zeilenumfang bei Druckdateien von den Standardfestlegungen abweichend gemacht werden können, wird diese Anweisung auch häufig für diese Dateien benutzt. Vorerst gibt es wenig Gründe, sie auch für andere Dateien einzusetzen.

Aufgabengruppe 28

1) Die Variablen B und C sind mit den Attributen DECIMAL FIXED(5,Ø) erklärt. Der für B einzulesende Wert ist in die Spalten 1 bis 3 der Lochkarten eines Kartenpaketes gelocht. Schreibe ein Programm, das alle Lochkarten liest und die Werte von B und C auf eine Zeile druckt. Der Wert von C ist dabei so auf der Druckzeile anzuordnen, daß zwischen der Einerstelle von B und der ersten von Null verschiedenen Ziffer bzw. des Minuszeichens von C gerade zwei Leerstellen verbleiben. Dezimalpunkte sollen nicht gedruckt werden. Der Wert von C errechnet sich aus der Relation

   C = 2B-1

2) Ändere die Lösung der Aufgabe 1) so ab, daß die Werte von B und C mit einem Dezimalpunkt gedruckt werden.

3) Welcher Wert wird durch das nachstehende Programm gedruckt?

```
A11Ø3:  PROCEDURE OPTIONS(MAIN) ;
            S = Ø ;
            T = 1 ;
        M:  F = 1 / (T**2+T) ;
            S = S + F ;
            IF F>1.ØE-6  THEN GOTO M ;
        END A11Ø3 ;
```

4) Welche Beziehung besteht zwischen X und Y, wenn die nachstehende Codierung vorliegt?

```
        .
        .
        IF  X>4  THEN GOTO M2 ;
        IF  X<Ø  THEN GOTO M3 ;
        X = 4*X - 9 ;
        GOTO M4 ;
M3:     Y = X - 9 ;
        GOTO M4 ;
M2:     Y = 3*X - 5 ;
M4:     Y = Y + 1 ;
        .
        .
```

5) Schreibe ein Programm, das x-Werte von Karten liest und anschließend x- und y-Werte druckt. Für die Berechnung der y-Werte gilt

$$y = \begin{cases} 3x^2 + 7 & \text{für } x > 8 \\ 2x^2 - 3 & \text{für } x < 1 \\ 2x & \text{für } 1 \leqq x \leqq 8 \end{cases}$$

6) Modifiziere das Programm zur Lösung der Aufgabe 5) wie folgt:

   a) Es sind 40 Wertepaare pro Seite zu drucken, auf jede Zeile ein Paar
   b) Auf jede Seite ist eine Überschriftszeile zu drucken; dabei soll der Buchstabe X über der letzten Stelle erscheinen, die die x-Werte einnehmen, der Buchstabe Y ebenso über der letzten Stelle der y-Werte.

12. Das Schreiben von PL/1-Programmen

12.1. ON-Bedingungen

Die ON-Anweisung besitzt die Form

ON bedingung [SNAP] oneinheit

Unter oneinheit ist entweder

SYSTEM ;

oder ein einfaches Statement, jedoch nicht ein PROCEDURE-, RETURN-, DECLARE-, DEFAULT-, END- oder FORMAT-Statement. ON- und auch IF-Anweisungen zählen nicht zu den einfachen Anweisungen; sie schließen als Verbundanweisungen mehrere Anweisungen ein.

Tritt die unter bedingung aufgeführte Bedingung auf, so wird die Oneinheit ausgeführt. Wird

SYSTEM ;

niedergeschrieben, so läuft eine sogenannte Standardsystemmaßnahme ab, kurz auch Standardmaßnahme oder Systemmaßnahme genannt. Bei der Codierung einer Leeranweisung unterbleibt beim Auftreten der aufgeführten Bedingung jede Maßnahme; die Programmablaufsteuerung kehrt zum nächstfolgenden Statement innerhalb des Programmes zurück. Bei Niederschrift der Option SNAP werden gewisse Nachweisinformationen über die Standarddruckdatei SYSPRINT protokolliert, die bei der Fehlersuche sehr nützlich sein können. Erst danach wird die Oneinheit ausgeführt.

Die Ausführung der ON-Anweisung verursacht nicht den Ablauf der Operationen in der Oneinheit selbst. Es wird vielmehr nur die Verbindung zur Oneinheit aufgebaut, so daß bei einem nachfolgenden Auftreten der spezifizierten Bedingung die Oneinheit ausgeführt werden kann. Nach dem durch das Auftreten der Bedingung verursachten Ablauf der Oneinheit erfolgt normalerweise die Rückkehr zu derjenigen Programmstelle im Programm, die unmittelbar auf das die Bedingung verursachende Statement folgt. Eine Ausnahme hiervon ergibt sich jedoch stets dann, wenn die Oneinheit aus einer unbedingten Sprunganweisung besteht. Das GOTO-Statement verhindert eben die direkte Rückkehr hinter die Programmstelle, an der die Bedingung zum Vorschein kam.

Eine komplette Liste aller Bedingungen, die durch die ON-Anweisung abgefangen werden können, befindet sich im Anhang E. Einige der Bedingungen wollen wir ihrer Wichtigkeit wegen jetzt kurz streifen.

1) EA-Bedingungen
   Bei EA-Operationen können folgende Bedingungen auftreten:

   a) ENDFILE(dateiname)
      Sie tritt auf, wenn ein Leseversuch unternommen wird, obwohl der letzte Satz der Datei bereits gelesen ist. Die Systemmaßnahme besteht darin, einen Kommentar auszugeben und die ERROR-Bedingung zu erzeugen.

   b) ENDPAGE(dateiname)
      Durch eine PUT-Anweisung wird versucht, eine Zeile zu drucken, obwohl das Blattende (Seitenende) bereits erreicht ist. Nach Rückkehr zum PUT-Statement, werden PAGE -, LINE- oder SKIP-Optionen bzw. Formatelemente, die die ENDPAGE-Bedingung verursacht haben, ignoriert. Die Systemmaßnahme veranlaßt einen Vorschub auf die nächste Druckseite. Unter dateiname kann nur der Name einer Druckdatei erscheinen.

c) TRANSMIT(dateiname)
Bei der Eingabe tritt diese Bedingung immer dann auf, wenn durch einen beständigen Fehler bei der Übertragung der Daten vom externen Medium in den Hauptspeicher einige gelesene Daten möglicherweise inkorrekt sind. Die Systemmaßnahme besteht in der Ausgabe eines Kommentars und im Setzen der ERROR-Bedingung.

Bei der Ausgabe kann diese Bedingung ebenfalls gesetzt werden und zwar dann, wenn ein Fehler bei der Übertragung der Daten vom Hauptspeicher auf das externe Medium nicht korrigiert werden kann.

Besteht die Oneinheit aus der Leeranweisung, so erfolgt bei der reihenweisen EA der Übergang zum nächsten Datenfeld.

d) UNDEFINEDFILE(dateiname).
Diese Bedingung wird gesetzt, wenn der Versuch unternommen wird, eine nicht verfügbare Datei zu benutzen. Die Systemmaßnahme besteht in der Ausgabe eines Kommentars und im Setzen der ERROR-Bedingung.

e) NAME(dateiname)
Diese Bedingung wird durch ein GET DATA-Statement gesetzt, das eine Eingabedatenreihe anspricht, in der der Name einer Variablen erscheint, die nicht in der Datenliste aufgeführt ist. Die Systemmaßnahme besteht in der Ausgabe eines Kommentars und in der Nichtbeachtung des die Bedingung verursachenden Feldes.

2) Rechenbedingungen
Zu den Rechenbedingungen zählen folgende Bedingungen:

a) CONVERSION
Diese Umwandlungsbedingung tritt immer dann auf, wenn eine Zeichenkette in einen anderen Datentyp umgewandelt werden soll, obwohl die Zeichenkette nicht den dafür geltenden Umwandlungsregeln genügt. Das sich durch diesen Umwandlungsversuch ergebende Resultat ist unbestimmt. Diese Bedingung kann nicht nur bei der Auswertung von Ausdrücken auftreten, sondern auch bei der Umwandlung von Eingabedaten, beispielsweise wenn ein durch ein B-Formatelement beschriebenes Feld andere Zeichen als 0 und 1 enthält. Auch bei Feldern mit lauter Leerzeichen, denen E-Formatelemente zugeordnet sind, oder bei numerischen Daten, die nichtnumerische Zeichen enthalten, kommt es zum Auftreten der CONVERSION-Bedingung. Eine Leeranweisung in die Oneinheit hineinzustellen, ist nicht möglich, da die normale Rückkehr darin besteht, daß die fragliche Umwandlung erneut versucht wird.

Die Systemmaßnahme besteht in der Ausgabe eines Kommentars und im Setzen der ERROR-Bedingung. Man beachte, daß diese Bedingung für jedes ungültige Zeichen innerhalb der angesprochenen Kette gesetzt wird.

b) FIXEDOVERFLOW
Das Ergebnis einer FIXED BINARY-Berechnung überschreitet 31 Bitstellen oder das Resultat einer FIXED DECIMAL-Berechnung erfordert mehr als 15 Dezimalstellen (diese Angaben haben für das System IBM /370 Gültigkeit). Das Ergebnis wird links abgeschnitten, d.h. daß die Stellen mit der höchsten Wertigkeit verloren gehen. Die Systemmaßnahme besteht in der Ausgabe eines Kommentars und im Setzen der ERROR-Bedingung.

c) OVERFLOW
Bei Gleitpunktrechnungen tritt ein Exponentenüberlauf auf. Das Ergebnis kann hierbei nicht vorausgesagt werden. Die Systemmaßnahme besteht in der Ausgabe eines Kommentars und im Setzen der ERROR-Bedingung.

d) UNDERFLOW
Bei Gleitpunktrechnungen kommt es zu einem Unterlauf beim Exponenten des Resultats. Das Ergebnis ist gleich Null. Die Systemmaßnahme besteht in der Ausgabe eines Kommentars und in der Fortsetzung der Berechnung und damit des Programmes. Diese Bedingung wird nicht gesetzt, wenn gleiche Zahlen voneinander subtrahiert werden.

e) ZERODIVIDE
Es wird der Versuch unternommen, durch Null zu dividieren, gleichgültig ob bei Festpunktrechnungen oder bei Gleitpunktrechnungen. Das Ergebnis ist nicht definiert. Die Systemmaßnahme besteht in der Ausgabe eines Kommentars und im Setzen der ERROR-Bedingung.

f) SIZE
Das zur Aufnahme des Ergebnisses einer Rechnung vorgesehene Feld ist nicht groß genug, um das gesamte Resultat aufnehmen zu können. Dadurch gehen die führenden Stellen des Ergebnisses verloren; das Ergebnis gilt als nicht definiert. Die Systemmaßnahme besteht in der Ausgabe eines Kommentars und im Setzen der ERROR-Bedingung.

3) Benutzerbedingungen
Vom Programmierer können willkürlich Bedingungen benannt und in ein Programm eingefügt werden. Sie sind in folgender Weise einzuführen:

CONDITION(bezeichnung)
Unter bezeichnung ist eine gültige externe Bezeichnung zu verstehen. Es können soviele Bedingungen gebildet werden, wie immer für erforderlich gehalten werden. Wird eine solche Bedingung forciert, so besteht die Systemmaßnahme in der Ausgabe eines Kommentars bei anschließender Weiterführung des Programmes.

4) Systembedingungen
Es gibt folgende Systembedingungen:

a) ERROR
Diese Bedingung wird dann gesetzt, wenn ein Fehler die Beendigung des Programmes erzwingt. Die Systemmaßnahme besteht darin, die FINISH-Bedingung zu setzen und einen Kommentar über die Standarddruckdatei SYSPRINT auszugeben.

b) FINISH
Diese Bedingung wird gesetzt, wenn das Programmende erreicht ist, d.h. im Normalfall durch Ausführung eines RETURN-, END- oder STOP-Statements. Diese Bedingung kann aber auch durch die ERROR-Bedingung hervorgerufen werden.

Durch die Systemmaßnahme wird die Beendigung des Programmes veranlaßt.

Mitunter wünscht der Programmierer, daß gewisse Bedingungen bei ihrem Auftreten nicht beachtet, andere dagegen beachtet werden. Die zu unterdrückenden Bedingungen werden ignoriert; es findet keine Verarbeitung der Oneinheiten statt, selbst wenn eine ON-Anweisung für die betreffende Bedingung ausgeführt wurde.

Wenn nicht anders vom Programmierer angegeben ist, befinden sich die folgenden Bedingungen zu Programmbeginn im wirksamen Status:

UNDERFLOW
OVERFLOW
ZERODIVIDE
CONVERSION
FIXEDOVERFLOW

Die folgenden fünf Bedingungen befinden sich dagegen zu Programmbeginn im unwirksamen Status:

```
SIZE
SUBSCRIPTRANGE
CHECK
STRINGRANGE
STRINGSIZE
```

Von diesen haben wir die letzten vier noch nicht besprochen. Alle anderen Bedingungen befinden sich ständig im wirksamen Status; sie können weder wirksam noch unwirksam gemacht werden.

Der Programmierer kann eine Bedingung dadurch in den wirksamen Status bringen, indem er den Bedingungsnamen in Klammern schreibt und ihn mit einem nachfolgenden Doppelpunkt versehen vor ein Statement setzt. Ist dieses Statement durch eine Marke markiert, so wird dieser sogenannte Bedingungspräfix noch vor dem Markenpräfix geschrieben, wie man aus folgendem Beispiel erkennen kann:

```
(SIZE): BK35Ø: Y = (X-3) * 4 ;
```

Soll eine Bedingung in den unwirksamen Status versetzt werden, so setzen wir einfach dem Bedingungsnamen die Silbe NO voran, wie man aus folgendem Beispiel ersehen kann:

```
(NOOVERFLOW,SIZE): IFA: Y = 3 + X ;
```

Durch diesen Bedingungspräfix wird einmal die Bedingung SIZE in den wirksamen Status und die Bedingung OVERFLOW in den unwirksamen Status gebracht. Wird ein Bedingungspräfix vor eine PROCEDURE-Vereinbarung (PROCEDURE-Statement) geschrieben, so bleibt er innerhalb der gesamten Prozedur gültig. Wenn ein solcher Präfix vor ein IF-Statement geschrieben wird, so hat er Wirksamkeit bis zum Ende des IF-Statements [1]. Wird er vor ein einfaches Statement gesetzt, so besitzt er nur Gültigkeit für das betreffende Statement. Ein einmal festgelegter Präfix kann dadurch für ein einziges Statement in das Gegenteil verkehrt werden. Beispielsweise bewirkt die nachfolgende Codierung, daß sich die OVERFLOW-Bedingung nur bei der mit dem entsprechenden Präfix versehenem Ergibtanweisung im wirksamen Status befindet.

```
(NOOVERFLOW): PR: PROCEDURE OPTIONS(MAIN) ;
                  :
                  :
          (OVERFLOW): Y = 76.4 * (x+2-3**X) ;
```

Die Menge der Statements, die durch eine ON-Bedingung ausgeführt bzw. angesprungen werden, wenn es zu einer entsprechenden Programmunterbrechung kommt, sollte so festgelegt werden, daß es zu einem sinnvollen Programmweiterlauf kommt, d.h. die Menge dieser Statements sollte nur dann zu der Programmstelle zurückgeführt werden, die die Unterbrechung veranlaßte, wenn diese Statements so aufgebaut werden konnten, daß sie eine Korrektur des Fehlers ermöglichen.

Das Statement

```
SIGNAL bedingung ;
```

1) Gilt jedoch nicht für in IF-Statements eingeschlossene Blöcke oder Gruppen von Statements.

bewirkt, daß sofort die angegebene Bedingung auftritt. Dieses Statement kann an einer beliebigen Programmstelle niedergeschrieben werden. Eine solche Codierung kann äußerst nützlich sein, wenn man ein recht kompliziertes Programm auf logische Irrtümer überprüft.

Ein Programmierer kann darüberhinaus auch seine eigenen Bedingungen festlegen, in dem er schreibt

ON CONDITION(bezeichnung) oneinheit

Das Auftreten einer solchen Bedingung kann selbstverständlich nur durch die SIGNAL-Anweisung hervorgerufen werden. Als Standardsystemmaßnahme ist für das Auftreten dieser Bedingung die Ausgabe eines Kommentars und die anschließende Fortsetzung des Programmes festgelegt.

Mit den ON-Bedingungen verfügen wir in PL/1 über ein Mittel, dessen wir uns sehr wirkungsvoll bedienen können, wenn es darum geht, ungewöhnliche Programmbedingungen und die dadurch hervorgerufenen Unterbrechungen in den Griff zu bekommen, ohne daß das Programm beim Nichtauftreten dieser Bedingungen verlangsamt wird. In den meisten Fällen wird es der Programmierer vorziehen, daß die Standard-Systemmaßnahmen für alle Bedingungen eingeleitet werden, d.h. er wird auf die Codierung von ON-Statements und damit auf eigene Maßnahmen verzichten. Zum Schluß sei betont, daß ON-Bedingungen durchaus nützlich bei der Fehlersuche in Programmen sein können.

12.2. Abkürzungen von Schlüsselwörtern

Wir haben schon eine Reihe von PL/1-Statements behandelt und alle diese Statements mit Ausnahme der Ergibtanweisung zeichneten sich dadurch aus, daß sie durch ein Leitwort eingeleitet wurden, dem Bezeichnungen folgten. Ein solches Leitwort ist ein Schlüsselwort der Sprache. Es legt fest, welche Funktion (Operation, Operationsfolge) ausgeführt werden soll, es bestimmt damit den Statementtyp. Das Ergibtstatement unterscheidet sich insofern von den anderen Statements, als daß das Ergibtzeichen, welches gewissermaßen die Rolle eines Schlüsselwortes übernommen hat, erst den Bezeichnungen von einer oder mehreren Variablen nachfolgt.

Sieht man von der Ergibtanweisung ab, so können wir alle Statements in eine allgemeine Form bringen, nämlich in die Form

(b_1,b_2,$\cdots$,b_k): m_1,m_2,$\cdots$,m_n: leitwort bezeichnungen ;

Hierbei bedeuten die b_i die Namen von Bedingungen und die m_j sind die Namen von Marken. Den Bezeichnungen können, wenn notwendig, noch Attribute folgen; ob diese vorhanden sind, hängt ausschließlich vom Typ des Statements ab. Beispielsweise stehen im DECLARE-Statement solche Attribute wie FIXED, DECIMAL, CHARACTER usw. Im GOTO-Statement andererseits ist die Angabe eines Attributes falsch. Jedes Statement wird durch ein Semikolon abgeschlossen.

Da einige der Leitwörter und Attribute, aber auch die Namen einiger Bedingungen häufig gebraucht werden, können für einige besonders lange Bezeichnungen bestimmte vorgeschriebene Abkürzungen verwendet werden. An Stelle von DECLARE kann z.B. DCL eingesetzt werden. Diese Abkürzungen können zusammen mit den vollen Bezeichnungen gleichzeitig in einem Programm benutzt werden; die Kompilierer können unterschiedslos beide Formen interpretieren. Eine vollständige Liste aller Abkürzungen befindet sich im Anhang E.

Es sollen nunmehr noch einige Abkürzungen zusammengestellt werden, die wir für Bezeichnungen verwenden dürfen, die wir bereits besprochen haben (Abb. 12.1.).

Bezeichnung	Abkürzung
PROCEDURE	PROC
DECLARE	DCL
DECIMAL	DEC
BINARY	BIN
COMPLEX	CPLX
CHARACTER	CHAR
COLUMN	COL
OVERFLOW	OFL
UNDERFLOW	UFL
FIXEDOVERFLOW	FOFL
ZERODIVIDE	ZDIV
CONVERSION	CONV
ENVIRONMENT	ENV

Abb. 12.1. Abkürzungen von Schlüsselwörtern

12.3. Anfangswerte

Wenn wir in einem DECLARE-Statement die Attribute für eine Variable festlegen, so können wir dieser auch einen Anfangswert zuweisen, d.h. einen Wert, der auf dem ihr zugeordneten Speicherplatz bereits beim Laden des Programmes in den Hauptspeicher gespeichert wird. Wir brauchen nur das Attribut

INITIAL(wert)

niederzuschreiben. An Stelle von INITIAL darf auch die Abkürzung INIT codiert werden. Betrachten wir zur näheren Erläuterung ein Beispiel (Abb. 12.2.).

```
DCL   A DECIMAL FIXED(5,Ø) INIT(14),
      B BINARY FIXED(15) INITIAL(-4),
      C COMPLEX FLOAT(6) INIT(3+4I),
      D CHAR(3) INITIAL('WXYZ') ;
```

Dieses Beispiel ist äquivalent dem folgenden Programmteil:

```
DCL   A DECIMAL FIXED(5,Ø),
      B BINARY FIXED(15),
      C COMPLEX FLOAT(6),
      D CHARACTER(3) ;
A = 14 ;
B = -4 ;
C = 3 + 4I ;
D = 'WXYZ' ;
```

Abb. 12.2. Festlegung von Anfangswerten

Mit dieser Auflösung erübrigt sich jede weitere Erklärung. Der Unterschied zwischen beiden Codierungen besteht darin, daß bei Festlegung eines Anfangswertes dieser bereits beim Laden des Programmes mit geladen wird, während bei Ergibtanweisungen die Werte der Variablen erst bei der Programmausführung zugewiesen werden.

Anfangswerte können genau so niedergeschrieben werden wie bei einer Ergibtanweisung. Der vier Zeichen umfassende Anfangswert für D wird infolgedessen an die Attribute von D angepaßt; somit wird D auf den Wert 'WXY' gesetzt.

12.4. Die Zusammenfassung von Attributen

Wenn mehrere Variablen die gleichen Attribute aufweisen sollen, so ist es gestattet, die Attribute für die Variablen in einer Zusammenfassung zu erklären. Hierzu müssen die Variablennamen zu einer Liste vereinigt werden, die von einem Klammernpaar umgeben ist. Die gemeinsamen Attribute werden dann rechts von der schließenden Klammer niedergeschrieben. Die Zusammenfassung von Attributen kann auch verschachtelt erfolgen.

Verfolgen wir ein Beispiel (Abb. 12.3.).

Zusammenfassung von Attributen

```
DCL  ((A FIXED(4,3),
       B FIXED(7,2),
       C FLOAT(6)) DECIMAL,
      D BINARY FIXED(15)) INIT(Ø) ;
```

Äquivalente Erklärungen

```
DCL  A DECIMAL FIXED(4,3) INITIAL(Ø),
     B DECIMAL FIXED(72,) INITIAL(Ø),
     C DECIMAL FLOAT(6) INITIAL(Ø),
     D BINARY FIXED(15) INITIAL(Ø) ;
```

Abb. 12.3. Zusammenfassung von Attributen

12.5. Eingefügte Funktionen

Bestimmte Berechnungen, wie die Bestimmung von Quadratwurzeln, Logarithmen, Absolutwerten usw., werden so häufig benötigt, daß spezielle Vorkehrungen für den PL/1-Programmierer getroffen wurden, die es ihm gestatten, derartige Funktionen in sein Programm direkt aufzunehmen. Der Programmierer braucht z. B. nur

```
Y = SQRT(X) ;
```

zu codieren und der Kompilierer erkennt SQRT als den Namen einer speziellen Funktion. Daraus resultiert, daß eine Maschineninstruktionsmenge, die zur Berechnung von Quadratwurzeln dient, automatisch in das Programm eingefügt wird. Bei der Wurzelfunktion kann das Argument irgendein nichtnegativer Ausdruck sein. Da die Wurzelfunktion auch für komplexe Argumente definiert ist, können wir sogar diese Funktion verwenden, wenn ein komplexes Argument vorliegt. Bei komplexen Argumenten erhalten wir als Funktionswert den Hauptwert der Potenz mit dem Argument als Basis und dem Exponenten Ø.5.

Wie bei jeder Ergibtanweisung wird das Resultat, d.h. hier der Funktionswert, so umgewandelt, daß es den Attributen von Y gerecht wird. Als Argument kann, wie schon erwähnt, irgendein Ausdruck verwendet werden. Funktionen können in einem Programm überall dort codiert werden, wo auch ein Ausdruck niedergeschrieben werden kann.

Eine Übersicht über die meisten der in PL/1 eingefügten Funktionen wird im Anhang E gegeben. Wir wollen nunmehr ein paar von ihnen betrachten.

LOG(x) berechnet den natürlichen Logarithmus von x. Als Resultat ergibt sich ein Gleitpunktwert mit einer dem Argument entsprechenden Genauigkeit. In gleicher Weise wird durch LOG1Ø(x) der dekadische Logarithmus von x errechnet.

Wird das Argument im Bogenmaß eingesetzt, so berechnet SIN(x) den Sinuswert von x, liegt dagegen das Argument im Gradmaß vor, so wird die Funktion SIND(x) einzusetzen sein.

Die Funktion

$$\text{MAX}(x_1, x_2, \cdots, x_n)$$

bestimmt den Maximalwert aller Argumente x_i mit $i=1,2,\cdots,n$. Das Resultat erscheint bei Typengleichheit der Argumente in derselben Form, in der die Argumente vorliegen. Unterscheiden sich die Argumente in ihren Attributen, so ergibt sich für den Funktionswert das Attribut FLOAT, wenn sowohl FIXED als auch FLOAT auftreten, BINARY, wenn sowohl BINARY als auch DECIMAL auftreten. Die Genauigkeit des Ergebnisses entspricht der größten Genauigkeit, die unter allen Argumenten vorkommt. Die Argumente der Funktion MAX müssen sämtlich das Attribut REAL aufweisen, da für komplexe Zahlen der Begriff „am größten" nicht definiert ist.

Die Funktion

```
HIGH(n),
```

bei der das Argument eine ganzzahlige dezimale Konstante ist, bewirkt, daß eine Zeichenkette der Länge n erzeugt wird, deren einzelne Zeichen die sedezimale Verschlüsselung FF [1] aufweisen, d.h. ein achtstelliges Bitmuster mit lauter Einsen, folglich das sortiermäßig höchste Zeichen. Das Gegenteil hierzu bildet die eingefügte Funktion

```
LOW(n),
```

die eine Zeichenkette der Länge n erzeugt, wobei die einzelnen Zeichen die sedezimale Verschlüsselung ØØ [1] aufweisen.

Eine Funktion kann durchaus als Argument einer anderen Funktion dienen. Zum Beispiel brauchen wir zur Berechnung der Quadratwurzel aus der Summe von e^x und e^{-x} nur die Ergibtanweisung

```
Y = SQRT (EXP(X)+EXP(-X)) ;
```

niederzuschreiben. Am Rande sei notiert, daß die Argumente von Funktionen stets in Klammern eingeschlossen sind. Bei mehreren Argumenten ist selbstverständlich die Argumentenliste in Klammern einzuschließen.

Da die Namen der eingefügten Funktionen nicht reservierte Namen sind, sondern durchaus auch als Namen von Variablen benutzt werden können, ist es mitunter möglich, daß die verschiedenen Verwendungen des gleichen Namens Verwirrung stiften. Der Name von eingefügten Funktionen kann deshalb grundsätzlich mit dem Attribut BUILTIN erklärt werden. Z.B. können wir codieren:

```
DCL   SQRT BUILTIN ;
```

Wird keine Erklärung vorgenommen, so wird angenommen, daß die eingefügte Funktion gemeint ist, vorausgesetzt, dem Namen folgen Klammern. Nun sind freilich einige ein-

1) Es ist der EBCDI-Code vorausgesetzt

gefügte Funktionen ohne Argumente aufzurufen. Bei diesen Funktionen muß dann der Programmierer entweder eine Vereinbarung mit dem Attribut BUILTIN abgeben oder aber er läßt dem Funktionsnamen eine leere Argumentenliste folgen. So wird z.B. in der Ergibtanweisung

```
ZK = DATE() ;
```

der Variablen ZK eine Zeichenkette der Länge 6 zugewiesen, die gleich dem augenblicklichen Datum in der Form jjmmtt [1] ist. Die eingefügte Funktion DATE besitzt nämlich diesen genannten Funktionswert; ZK ist deshalb zweckmäßigerweise mit

```
DCL  ZK CHAR(6) ;
```

zu vereinbaren.

Einige eingefügte Funktionen sind speziell für die Verwendung in Oneinheiten (siehe Abschnitt 12.1.) geschaffen worden. So gibt z.B. ONSOURCE einen Funktionswert zurück, der gleich der Zeichenkette ist, die aus dem Inhalt desjenigen Feldes besteht, das bei seiner Verarbeitung zum Auftreten der Bedingung CONVERSION führte.

12.6. Indizes

Angenommen, wir haben ein Polynom der Form

$$y = a_0 + a_1x + a_2x^2$$

zu berechnen. Wir können es umformen in

$$y = a_0 + x(a_1 + a_2x)$$

In dieser Form sind nur noch zwei Multiplikationen erforderlich. Nehmen wir an, daß die Koeffizienten a_0, a_1 und a_2 in Gleitpunktdarstellung vorliegen und daß wir die y-Werte für verschiedene x-Werte berechnen sollen, so können wir ein Programm abfassen, das die in Abb. 12.4. vorgestellte Form annimmt.

```
POLBER:  PROCEDURE OPTIONS(MAIN) ;
           DECLARE  A(3) FLOAT,
                    X FLOAT, Y FLOAT ;
           GET LIST (A(1),A(2),A(3)) ;
           X = -1Ø ;
SCHL:      Y = A(1) + X*(A(2) + X*A(3)) ;
           PUT LIST (X,Y) SKIP(1) ;
           X = X + 1 ;
           IF  X<=1Ø  THEN GOTO SCHL ;
         END POLBER ;
```

Abb. 12.4. Programm zur Berechnung von Polynomen

Bei diesem Beispiel haben wir die Werte für die drei Koeffizienten a_0,a_1 und a_3 einzulesen. Da diese gemeinsame Eigenschaften haben, liegt es nahe, sie mit A_1, A_2 und A_3 zu bezeichnen. Wir können aber nur einzeilige Bezeichnungen niederschreiben und danach ab-

1) Es bedeuten:

j	Jahr
m	Monat
t	Tag

lochen, d. h. also wir müssen auf die Indexschreibweise verzichten. Es ist deshalb festgelegt worden, daß Indizes in Klammern eingeschlossen werden und dem Namen angehängt werden.

Um von dieser Schreibweise Gebrauch machen zu können, müssen wir den maximalen Umfang einer derartigen Bereichsvariablen in einer DECLARE-Vereinbarung festlegen, d. h. aber, den größtmöglichen Wert angeben, den die Indizes annehmen können. Dadurch wird der Kompilierer in die Lage versetzt, genügend Speicherplatz im Objektprogramm für alle Elemente des Bereiches (Vektors) zu reservieren. Der Maximalwert, der in einer Bereichserklärung angegeben werden kann, beträgt 32767 (beim System IBM /370). Nach Einfügung dieser Erklärung können wir uns auf die Elemente $A_1, \cdots$ einfach dadurch beziehen, daß wir den Variablennamen A niederschreiben und in Klammern den Index folgen lassen. Jeder Index kann als Ausdruck angegeben werden. Beim Ablauf des Programmes wird zunächst der Indexausdruck berechnet und in eine duale ganze Zahl in Festpunktdarstellung mit umgewandelt, wobei dieselbe den in der DECLARE-Vereinbarung festgehaltenen Maximalwert nicht überschreiten darf.
In unserem Beispiel (Abb. 12.4.) darf also der Indexausdruck nur die Werte 1, 2 und 3 annehmen. Das Beispiel selbst kann relativ leicht verfolgt werden, da alle ins Programm eingefügten Indexausdrücke nur Konstanten sind.

Es darf mehr als ein einziger Index verwendet werden. So dürfen wir z. B. zur Beschreibung des Lösungsverfahrens für ein System von zwei linearen Gleichungen mit zwei Unbekannten

I. $a_{11}x + a_{12}y = b_1$

II. $a_{21}x + a_{22}y = b_2$

die Bereichsvariable A mit zwei Dimensionen und die Bereichsvariable B mit einer Dimension erklären, indem wir folgendes DECLARE-Statement verwenden:

```
DCL  A(2,2) FLOAT, B(2) FLOAT ;
```

Wenn der Kompilierer die vier für A erforderlichen Speicherplätze einrichtet, so ordnet er sie für das Objektprogramm in folgender Reihenfolge an:

A(1, 1)
A(1, 2)
A(2, 1)
A(2, 2)

Dieses vom Kompilierer vorgenommene Anordnungsverfahren kann man sich leicht merken, wenn man sich vergegenwärtigt, daß sich der am weitesten rechts stehende Index am häufigsten und der am weitesten links stehende am wenigsten verändert.

Wie wir bereits stillschweigend eingeführt haben, müssen zwei oder mehr Indizes voneinander durch Kommas getrennt werden.

Werden bei Eingabe/Ausgabeanweisungen nur die Bereichsnamen, jedoch keine Indizes angeführt, so wird damit erreicht, daß der gesamte Bereich gelesen bzw. geschrieben wird.

Beispielsweise wird durch die Anweisung

```
GET LIST(A) ;
```

bewirkt, daß vier Werte vom externen Medium gelesen werden und der Reihe nach den Bereichselementen (A(1, 1), A(1, 2), A(2, 1) und A(2, 2) zugewiesen werden. Dagegen veranlaßt das Statement

```
GET LIST (A(1,1),A(2,1)) ;
```

das Einlesen von nur zwei Werten, die den Bereichselementen A(1, 1) und A(2, 1) zugewiesen werden.

Der Programmierer kann sich in seinem Programm auch auf einen Unterbereich von Bereichen beziehen, indem er ein Sternzeichen für diejenigen Indizes einträgt, die variieren sollen. So erreicht man durch das Statement

```
GET LIST (A(1,*)) ;
```

daß Werte für alle diejenigen Bereichselemente von A eingelesen werden, deren erster Index gleich 1 ist, also für A(1, 1) und A(1, 2). Daraus folgt sofort, daß für den gesamten Bereich zwei äquivalente Schreibweisen existieren: Die eine zeichnet sich dadurch aus, daß man nur den Namen des Bereiches niederschreibt, die andere dadurch, daß man dem Bereichsnamen eine Klammer folgen läßt und jede Indexposition durch ein Sternzeichen ausfüllt. In unserem Fall sind also A und A(*, *) völlig gleichwertig. Es sei betont, daß ein Unterbereich stets selbst wieder ein Bereich ist.

Bereichsnamen dürfen auch in Ergibtanweisungen verwendet werden. Wird so verfahren, so werden Rechenoperationen zwischen den einander entsprechenden Bereichselementen durchgeführt. Somit wird durch

```
DCL  C(2,3), D(2,3), E(2,3) ;
  .
  .
  .
E = C + 2*D + 4 ;
```

erreicht, daß jedes Element des Bereiches E ersetzt wird durch die Summe aus dem entsprechenden Bereichselement von C und dem Zweifachen des entsprechenden Bereichselementes von D, vermehrt um 4. Wir wollen dieses Beispiel an Hand zweier willkürlich ausgewählter Matrizen im einzelnen verfolgen:

$$C = \begin{bmatrix} 7 & 9 & 6 \\ 5 & -2 & \emptyset \end{bmatrix} \qquad D = \begin{bmatrix} 2 & 3 & 5 \\ -1 & 4 & -2 \end{bmatrix}$$

$$E = \begin{bmatrix} 7+2\cdot 2+4 & 9+2\cdot 3+4 & 6+2\cdot 5+4 \\ 5+2\cdot(-1)+4 & -2+2\cdot 4+4 & \emptyset+2\cdot(-2)+4 \end{bmatrix}$$

$$E = \begin{bmatrix} 15 & 18 & 2\emptyset \\ 7 & 1\emptyset & \emptyset \end{bmatrix}$$

Es sei hier unbedingt darauf verwiesen, daß die skalare Größe 4 zu jedem Bereichselement addiert wird. Schreibt man in PL/1 die Ergibtanweisung

```
E = C*D + 4 ;
```

nieder, so wird durch diese bewirkt, daß jedes Element von C mit jedem Element von D multipliziert und das Produkt um 4 vermehrt wird. Damit ist aber gesagt, daß die Matrixmultplikation in anderer Form codiert werden muß.

Wir müssen Vorsicht walten lassen, wenn wir Ergibtanweisungen niederschreiben, in denen ein Bereichselement an den gesamten Bereich oder einen Teil desselben gekoppelt ist. Betrachten wir z. B. hierzu die Ergibtanweisung

```
D = D * D(1,1) ;
```

und verfolgen wir ihre Auflösung, so erhalten wir eine Serie von skalaren Ergibtanweisungen

```
D(1,1) = D(1,1) * D(1,1) ;
D(1,2) = D(1,2) * D(1,1) ;
```

usw.

Durch den ersten Schritt wird bereits der ursprüngliche Wert von D(1,1) geändert, so daß der zur Berechnung der Werte der folgenden Bereichselemente herangezogene Wert von D(1,1) ein anderer ist als der beim ersten Schritt benutzte.

In der Regel besitzen die Indizes die Attribute BINARY FIXED(15,Ø) und sind gleich den natürlichen Zahlen $1,2,\cdots,n$. PL/1 erlaubt darüberhinaus auch den Gebrauch des Index Ø sowie von negativen ganzzahligen Indizes. Voraussetzung ist freilich eine entsprechende Definition im DECLARE-Statement. Derartige Dimensionsfestlegungen für Indizes sollen an einem Beispiel erläutert werden:

```
DECLARE  F(Ø:2), G(-3:1,2:4) ;
```

F wird hierdurch als Bereichsvariable mit einer Dimension festgelegt, deren untere Grenze 0 und deren obere Grenze 2 ist; G hingegen weist zwei Dimensionen mit den unteren Grenzen -3 und 2 sowie mit den oberen Grenzen 1 und 4 auf. Zwischen unterer und oberer Grenze jeder Dimension ist also ein Doppelpunkt zu schreiben; ist die unterer Grenze gleich 1, so darf die Angabe der unteren Grenze weggelassen werden, wie wir es bereits weiter oben gesehen haben. Aus dieser Festlegung heraus können wir ohne weiteres auch herleiten, daß der Bereich F aus drei und der Bereich G aus 5·3=15 Elementen besteht. Schreibt man bei einem anderen Fall z.B.

```
DCL  H(Ø:3,2) ;
```

so ist durch diese Erklärung der Bereich H eindeutig festgelegt: Er enthält acht Elemente in der Reihenfolge H(0,1), H(0,2), H(1,1), H(1,2), H(2,1), H(2,2), H(3,1) und H(3,2). Zusammenfassend soll noch einmal festgestellt werden, daß zwischen den einzelnen Dimensionen ein Komma stehen muß und daß bei Angabe von nur einer Grenze die untere Grenze grundsätzlich zu 1 angenommen wird.

Verwenden wir den Index 0, so können wir das in Abb. 12.4. vorgestellte Beispiel zur Berechnung der Werte des Polynoms $y = a_0 + a_1x + a_2x^2$ so codieren, wie es die Abb. 12.5. zeigt.

```
POLBER:  PROCEDURE OPTIONS(MAIN) ;
           DECLARE  (A(Ø:2),X,Y) FLOAT ;
           GET LIST(A) ;
           X = -1Ø ;
SCHL:      Y = A(Ø) + X*(A(1)+X*A(2)) ;
           PUT LIST(X,Y) SKIP(1) ;
           X = X + 1 ;
           IF  X<=1Ø  THEN GOTO SCHL ;
         END POLBER ;
```

Abb. 12.5. Programm zur Berechnung von Polynomen (modifizierte Fassung des Programms in Abb. 12.4.)

Auf einen Umstand sollte noch besonders hingewiesen werden: Durch die in PL/1 festgelegte Schreibweise der Indizes ist es leicht möglich, Funktionen mit indizierten Variablen zu verwechseln. Es möge z.B. folgender Auszug aus einem Programm vorliegen:

```
  .
  .
DCL  SQRT(4) ;
  .
  .
Y = SQRT(X) ;
  .
  .
```

Hier legt man sich bei der Ergibtanweisung unwillkürlich die Frage vor, ob die Wurzel aus X oder das X-te Bereichselement des Bereiches SQRT gemeint ist. Sprachlich gesehen ist durch die DCL-Vereinbarung eine eindeutige Festlegung getroffen: Es gilt selbstverständlich das letztere. Um persönlichen Verwechslungen dieser Art vorzubeugen, sollte es ein Programmierer tunlichst vermeiden, Funktionsnamen als Bereichsnamen einzuführen.

Auch den Elementen von Bereichen können Anfangswerte zugewiesen werden. Hierzu brauchen wir nur hinter das Schlüsselwort INITIAL eine von Klammern umgebene Werteliste zu stellen. Wie üblich bei Listen sind zwischen zwei aufeinanderfolgende Werte Kommas zu stellen. Verfolgen wir ein erstes Beispiel:

```
DCL  J DECIMAL FIXED(3,1) INIT(4.2),
     K(4) DECIMAL FIXED(3,1) INIT(Ø,1,3.2,-8.3) ;
```

Es ist gestattet, einige Elemente eines Bereiches von der Zuweisung von Anfangswerten auszuschließen. Das Fehlen eines Anfangswertes wird durch ein Sternzeichen angezeigt. Somit wird durch die Vereinbarung

```
DCL  L(4) DECIMAL FIXED(5,1) INIT(Ø,1,*,-8.2) ;
```

bestätigt, daß das Bereichselement L(3) ohne Anfangswert bleiben soll.

Soll mehreren aufeinanderfolgenden Elementen eines Bereiches der gleiche Anfangswert zugewiesen werden, so darf ein Wiederholungsfaktor vor diesen Wert geschrieben werden. Somit zeigt die Vereinbarung

```
DCL  D(Ø:3,2) DECIMAL FIXED(2,1)
              INITIAL (1,(4)Ø,2,(2)Ø) ;
```

an, daß D(Ø,1) auf den Anfangswert 1 gesetzt wird; der Anfangswert Ø wird den vier aufeinanderfolgenden Elementen D(Ø,2), D(1,1), D(1,2) und D(2,1) zugewiesen. Das nächstfolgende Bereichselement D(2,2) wird auf 2 gesetzt und die restlichen beiden, nämlich D(3,1) und D(3,2) auf Ø.

Wir sollten stets daran denken, daß die Wiederholungsfaktoren in Klammern zu schreiben sind, die Anfangswerte hingegen nicht. Sternzeichen zeigen fehlende Anfangswerte an. Außerdem wäre noch zu bemerken, daß Wiederholungsfaktoren ineinander geschachtelt werden dürfen. Hierfür ein Beispiel zu bringen, ist sicher recht lehrreich. Die Vereinbarung

```
DCL  M(7) INIT (-4,*,1,*,1,3,3) ;
```

ist äquivalent der Vereinbarung

Überschreitet die Anzahl der aufgeführten Anfangswerte die Zahl der Bereichselemente, so bleiben die überzähligen Anfangswerte unbeachtet. Umgekehrt ist die Zahl der Bereichselemente größer als die Zahl der Anfangswerte, so werden den restlichen Bereichselementen keine Anfangswerte zugewiesen; es ist so, als ob für sie Sternzeichen notiert wären. Die Anfangswerte werden den Bereichselementen in der Reihenfolge zugewiesen, in der sie im Hauptspeicher angeordnet sind.

Bei Kettenvariablen könnte es zu einer Verwechslung zwischen den bei ihnen möglichen zwei verschiedenen Wiederholungsfaktoren kommen, nämlich zwischen dem Wiederholungsfaktor für in der Kette stehende Zeichen und dem Wiederholungsfaktor für die Initialisierung. Deshalb ist festgelegt, daß grundsätzlich alle beiden Wiederholungsfaktoren geschrieben werden müssen, wenn ein Wiederholungsfaktor für die Initialisierung benötigt wird, auch

wenn der Kettenwiederholungsfaktor gleich 1 ist, d.h. keine Wiederholung von Kettenzeichen vorgesehen ist. Der Wiederholungsfaktor für die Initialisierung muß dabei zuerst niedergeschrieben werden. Man verfolge das nachstehende Beispiel:

```
DCL   BU CHAR(5). INIT((5)'G'),
      BV(4) CHAR(8) INIT((4)(1)'ABCDEFGH'),
      RS(3) CHAR(2) INIT ((4)'HJ') ;
```

Diese Vereinbarung veranlaßt die Zuweisung folgender Anfangswerte:

```
  BU   ←——   'GGGGG'
BV(1)  ←——   'ABCDEFGH'
BV(2)  ←——   'ABCDEFGH'
BV(3)  ←——   'ABCDEFGH'
BV(4)  ←——   'ABCDEFGH'
RS(1)  ←——   'HJ'
```

Die Elemente RS(2) und RS(3) bleiben ohne Anfangswerte, da der Wiederholungsfaktor (4) als Wiederholungsfaktor für die Zeichenkette 'HJ' interpretiert wird, somit die acht Zeichen umfassende Zeichenkettenkonstante 'HJHJHJHJ' ergebend. Wegen des Längenattributes (2) von RS werden jedoch nur die beiden linksbündigen Zeichen H und J dem Bereichselement RS(1) als Anfangswert zugewiesen.

Als Wiederholungsfaktoren für die Initialisierung werden gewöhnlich positive Konstanten benutzt, doch ist es erlaubt, dafür auch beliebige gültige Ausdrücke zu verwenden.

Ein Faktor 0 bzw. ein negativer Faktor bewirkt dabei keine Initialisierung. Durch

```
INIT(2, (2*N+3)4)
```

wird somit erreicht, daß zunächst der Wert 2 und anschließend (2*n+3)-mal der Wert 4 als Anfangswert zugewiesen wird. Zuvor muß jedoch der Variablen N selbst ein Wert zugewiesen sein, beispielsweise durch eine Ergibt- oder eine Eingabeanweisung in einem übergeordneten Programmteil oder dadurch, daß N als Parameter erklärt, der beim Aufruf mit einem Wert versorgt wird.

12.7. Markenvariablen

Angenommen, wir müssen in einem Programm den Wert der Variablen V prüfen. Besitzt V den Wert 1, so sollen die Routinen durchlaufen werden, deren jeweils erstes Statement durch die Marken A bzw. B bzw. C gekennzeichnet ist. Ist V dagegen gleich 2, so sollen die durch A, B und D markierten Routinen ausgeführt werden, und bei V=3 sollen schließlich die mit E, B und C markierten Routinen ausgeführt werden. Wir können das durch die folgende Codierung erreichen:

```
 A:   ...
      GOTO B ;
 B:   ...
      GOTO HA1 ;
 C:   ...
      GOTO HA2 ;
 D:   ...
      GOTO HA2 ;
 E:   ...
      GOTO B ;
HAØ:  IF  V=1 | V=2  THEN GOTO A ;
                     ELSE GOTO E ;
HA1:  IF  V=2  THEN GOTO D ;
                    ELSE GOTO C ;
HA2:  ...
```

In vielen Fällen ist es jedoch nicht zweckmäßig, den Wert einer solchen Variablen wie V unverändert zu lassen, um ihn im weiteren Programmverlauf noch prüfen zu können. In einem solchen Fall möchten wir uns am liebsten auf einen Schalter beziehen, den wir der Logik des Programmes entsprechend setzen können, wodurch wir uns die Anweisung

```
GOTO HA1 ;
```

ersparen. Ein solches Statement mit einem Schalter S würde es uns gestatten, das Statement am Ende der Routine B durch

```
IF  S  THEN GOTO D ;
       ELSE GOTO C ;
```

zu ersetzen. S müßte hierbei eine Bitkettenvariable mit der Länge 1 sein. Besitzt S dann den Wert '1'B, so würde eine Verzweigung nach D erfolgen, anderenfalls nach C. In HA0 müßten wir dann vor dem Verzweigen den Wert von V prüfen und aufgrund dieser Prüfung S den passenden Wert zuweisen.

In PL/1 können wir auch eine andere Methode wählen. Wir legen fest, daß HA1 eine Markenvariable ist. Dann müssen wir vor dem Statement mit der Marke HAØ eine IF-Anweisung einbauen, etwa

```
IF  V=2  THEN HA1=D ;
         ELSE HA1=C ;
```

Wir haben also in die THEN- bzw. ELSE-Klausel der IF-Anweisung Ergibtanweisungen eingefügt, die den „Wert" D bzw. C der Variablen HA1 zuweisen. Alle Markenvariablen, wie hier HA1, müssen in einem DECLARE-Statement als Markenvariable erklärt werden, indem für sie das Attribut LABEL angegeben wird. In unserem Fall lautet die Vereinbarung

```
DCL  HA1 LABEL ;
```

Einer Markenvariablen darf in einer Ergibtanweisung nur eine andere Markenvariable oder eine Markenkonstante zugewiesen werden. Wir wollen uns in diesem Zusammenhang daran erinnern, daß die vor einem Statement stehenden Marken Konstanten sind, in unserem Fall sind C und D derartige Markenkonstanten. Bei Vergleichen zwischen Markendaten sind allein nur die Operatoren „gleich" und „nicht gleich" zulässig. Ist ein Statement mit zwei oder mehr Markenkonstanten versehen, so gelten alle diese Markenkonstanten als gleich.

Die Markenvariablen können auch in einem Bereich organisiert sein, genau so wie die anderen Variablen auch. Die notwendige Information über die Anzahl und die Grenzen der Dimensionen müssen dann natürlich in die Erklärung mit aufgenommen werden. Somit können wir in einem Programm z.B. folgende Erklärung abgeben:

```
DCL  MAVAR(7) LABEL ;
```

Dann ist es uns möglich, daß wir uns auf MAVAR(2), MAVAR(J+4) usw. beziehen. Wir können also Statements verwenden, die z.B. wie folgt aussehen:

```
GOTO MAVAR(J-1) ;
IF  MAVAR(J)=HA2  THEN MAVAR(3)=HAØ ;
MAVAR(2) = B ;
```

Anfangswerte können Markenvariablen genau so zugewiesen werden wie anderen Variablen auch. Beispiele hierfür sind:

```
DCL  BH(3) LABEL INIT(B,C,HA2) ;
DCL  BK(3) LABEL INIT(*,D,*) ;
```

Die letztere Erklärung weist BK(2) den Wert D zu, BK(1) und BK(3) erhalten hingegen keine Anfangswerte zugewiesen.

Eine Markenvariable kann niemals in den Markenpräfix eines beliebigen Statements gesetzt werden. Somit ist

```
          DCL  HA1 LABEL ;
          :
          :
          ...
HA1:
```

eine ungültige Folge von Statements. Wenn einem Bereich von Markenvariablen jedoch das Attribut INITIAL nicht zugeordnet ist, so können die einzelnen Elemente dieses Bereiches auf eine besondere Art initialisiert werden. Hierzu muß man den mit einem konstanten Index versehenen Elementnamen einfach in einem Markenpräfix vor ein Statement schreiben. Der konstante Index muß eine ganze Dezimalzahl sein und im Spektrum der bei der Erklärung des Markenbereiches festgelegten Dimensionsgrenzen liegen. Ein auf dieser Basis eingeführte Wert gilt als Anfangswert. Wenn wir uns unter diesem Aspekt die Statementfolge

```
                DCL  M(4) LABEL ;
                :
                :
        M(3):   ...
MM:     M(2):   ...
```

ansehen, so können wir schlußfolgern, daß Anfangswerte für M(2) und M(3) vorliegen. Für M(2) existiert ein äquivalenter Name, nämlich MM. Ein solcher fehlt augenscheinlich für M(3); das Element M(3) des Markenbereiches M besitzt gewissermaßen seinen eigenen Namen als Wert.

In einer Erklärung kann dem Attribut LABEL eine Liste von Markenkonstanten folgen. Beispielsweise könnte eine Vereinbarung mit einer solchen Erklärung das Aussehen

```
DCL  HA1 LABEL(D,C) ;
```

besitzen. Durch eine solche Festlegung wird der Wertevorrat der Markenvariablen, hier HA1, auf die in der Liste enthaltenen Markenkonstanten beschränkt. Nur diese Markenkonstanten dürfen dann innerhalb eines Programmes als Werte für die Markenvariable HA1 fungieren. Eine solche Festsetzung hat auch sicher Vorteile: Neben der klaren Gliederung des Programmes und der dem Programmierer auferlegten Selbstdisziplin ist zu beachten, daß der Kompilierer befähigt wird, eine wirkungsvollere Instruktionsfolge zu generieren. Die logische Ausführung eines Programmes bleibt von einer solchen Einschränkung des Wertevorrates jedoch unberührt.

12.8. Logische Operationen

Für arithmetische Operationen sind, wie könnte es auch anders sein, Daten des arithmetischen Typs besonders geeignet. Daten vom Zeichenkettentyp wiederum ermöglichen die besonders einfache Handhabung von Zeichenfolgen (Texten) beispielsweise bei der Speicherung, bei der Eingabe und Ausgabe usw. Bitketten schließlich sind geradezu prädestiniert, als Operanden in logische Operationen, wie „und", „oder" und „nicht" einzugehen.

Angenommen, es sind die folgenden Bitkettenvariablen erklärt:

```
DECLARE  BA BIT(4),
         BB BIT(4),
         BC BIT(8) ;
```

Außerdem mögen die folgenden Ergibtanweisungen ausgeführt worden sein:

```
BA = 'ØØ11'B ;
BB = 'ØØØ1'B ;
```

Betrachten wir zunächst die drei logischen Operationen im Rahmen von drei verschiedenen Ergibtanweisungen:

1) Und
   Durch die Ergibtanweisung

   ```
   BC = BA & BB ;
   ```

   wird BC der Wert 'ØØØ1ØØØØ'B zugewiesen.

2) Oder
   Durch die Ergibtanweisung

   ```
   BC = BA | BB ;
   ```

   wird BC der Wert 'ØØ11ØØØØ'B zugewiesen.

3) Nicht
   Durch die Ergibtanweisung

   ```
   BC = ¬BA ;
   ```

   wird BC der Wert '11ØØØØØØ'B zugewiesen. Hier ist zu beachten, daß BA eine vierstellige Zeichenkette darstellt, die bei der Zuweisung zu BC auf acht Bitstellen erweitert wird, indem rechts vier Nullen angehängt werden. Ebenso ergibt

   ```
   BC = ¬BB ;
   ```

   für BC den Wert '111ØØØØØ'B.

Gemeinsam bei allen drei logischen Operationen ist hier, daß alle Operanden vierstellige Bitketten sind. Das Resultat der logischen Operation ist deshalb ebenfalls eine vierstellige Bitkette. Da die zur Aufnahme des Ergebnisses vorgesehene Variable BC mit der Länge 8 erklärt ist, muß eine Anpassung des Resultates an die Attribute von BC erfolgen. Da alle Ketten linksbündig zugewiesen werden, wird nach rechts eine Ergänzung mit Nullen vorgenommen.

Die Verkettungsoperation kann ebenfalls auf Bitketten angewendet werden. Durch

```
BC = BA || BB ;
```

ergibt sich für BC der Wert 'ØØ11ØØØ1'B.

Aber nicht nur logische Operationen und Verkettungen können wir mit Bitketten ausführen, sondern auch arithmetische Operationen. Dabei müssen wir uns freilich vergegenwärtigen, daß die Bitketten erst in arithmetische Werte umgewandelt werden müssen, bevor es zur Ausführung der arithmetischen Operationen kommt. Doch verfolgen wir das an Beispielen mit unseren definierten Variablen BA, BB und BC. Wenden wir unser Augenmerk zunächst der Addition zu.

```
BC = BA + BB ;
```

Die beiden Bitketten werden jetzt als ganze Dualzahlen aufgefaßt; als Ergebnis erhalten wir jedoch BC='ØØØØØØØØ'B und nicht, wie wir vielleicht erwartet haben, ØØØØØ1ØØ. Um das Entstehen dieses Ergebnisses zu verfolgen, müssen wir die Schritte betrachten, die nacheinander ablaufen. Zunächst werden die Werte von BY und BB in eine ganze Dualzahl mit der Genauigkeit (15.0) umgewandelt. Danach können sie addiert werden; als Resultat bekommen wir die Dualzahl 0000|0000|0000|0100. Zum Schluß wird nun dieser Wert wieder in eine Bitkette umgewandelt, um den Attributen zu entsprechen, die für BC festgelegt sind. Bei dieser Umwandlung ergibt sich eine Bitkette der Länge 15; da für BC aber nur acht Bitstellen erklärt sind, werden die rechten sieben Stellen abgeschnitten und es ergibt sich deshalb für BC eine aus lauter Nullen bestehende Bitkette.

Codieren wir das Statement

```
BC=BA*BB;
```

unter Beibehaltung der oben aufgeführten Erklärungen, so erhalten wir als Ergebnis für BC ebenfalls eine aus lauter Nullen bestehende Bitkette. Bevor die Multiplikation ausgeführt wird, werden die Werte von BY und BB zunächst in 15stellige ganze Dualzahlen umgewandelt; diese werden miteinander multipliziert, wobei sich ein 31stelliges Produkt bildet. Dieses wird schließlich in eine ebenso lange Bitkette umgewandelt, wodurch sich nach dem Abschneiden der rechten 23 Bitstellen das schon erwähnte Resultat ergibt.

Das oben behandelte Beispiel legt uns eindeutig nahe, besonders auf die Vorgänge achtzugeben, die bei der Durchführung von arithmetischen Operationen mit Bitketten oder Zeichenketten ablaufen. Datentypumwandlungen sind und bleiben nun einmal kritisch in jeder Beziehung.

Was für ein Ergebnis finden wir, wenn wir bei arithmetischen Operationen mit Bitketten das Resultat einer arithmetischen Variablen zuweisen? Orientieren wir uns an der Ergibtanweisung

```
J = BA + BB ;
```

Für J liege keine explizite Erklärung vor, so daß die Variable J durch die Attribute BINARY FIXED(15) charakterisiert ist. Hier werden zunächst die Bitketten in ganze Dualzahlen in Festpunktdarstellung mit der Genauigkeit (31) umgewandelt, bevor die Addition ausgeführt wird. Damit wird aus der Bitkette von BA der arithmetische Wert 3 und aus der Bitkette von BB der Wert 1. Beide zusammen addiert führen zum Resultat 4, das dann J zugewiesen wird.

Wenn wir nun die Ergibtanweisung

```
BC = J ;
```

niederschreiben, so wird der Wert von J zunächst in eine fünfzehnstellige Bitkette umgewandelt. Diese wird nun BC zugewiesen. Da BC aber nur Platz für 8 Bits hat, wird die fünfzehnstellige Bitkette, die sich aus der Umwandlung von J ergab, linksbündig nach BC gestellt und die rechten sieben Bits, d.h. die Bits niedriger Ordnung, abgeschnitten. Somit nimmt BC den Wert 'ØØØØØØØØ'B an. Hätten wir jedoch für J die Erklärung

```
DCL  J BIN FIXED(8) ;
```

abgegeben und uns nicht auf die Standardgenauigkeit 15 gestützt, so wäre das J zugewiesene Resultat verschieden gegenüber dem ersten, wenn wir wiederum

```
BC = J ;
```

schreiben. Es ergibt sich nämlich dann 'ØØØØØ1ØØ'B. Lassen wir nun dieses Spiel mit Bitketten in arithmetischen Ausdrücken und Ergibtanweisungen. Wir hoffen, daß ein Lerneffekt

erzielt wurde, der uns zur Einsicht verhalf, daß arithmetische Operationen mit Bitketten nur mit Sorgfalt zu planen sind.

Die logischen Operationen dürfen in einem Ausdruck zusammen mit arithmetischen Operationen verwendet werden. Betrachten wir deshalb ein Beispiel (Abb. 12.6.).

```
        DECLARE  BA BIT(4),    BB BIT(4),
                 CA BINARY FIXED(31,Ø),
                 CC BINARY FIXED(31,Ø) ;
MA:     CA = 345 ;    CC=169 ;
MB:     CC = CC - 138 ;
        BA = 'Ø11Ø'B ;
        BB = 'ØØØ1'B ;
        IF  CA<CC+8ØØ & BA>BB  THEN GOTO MB ;
                               ELSE BB='ØØ1Ø'B ;
```

Abb. 12.6. Ausdrücke mit verschiedenen Arten von Operationen

Bei der Berechnung des Ausdruckes

```
CA<CC+8ØØ & BA>BB
```

wird zunächst die Addition CC+8ØØ ausgeführt, die als Resultat 1ØØ7 ergibt, was aber wiederum größer als 345 ist. Das Ergebnis des ersten Operanden für die logische Und-Verknüpfung ist somit wahr, d.h. '1'B. Der Vergleich von BA mit BB wird von links nach rechts Bit für Bit ausgeführt, wodurch sich ebenfalls das Ergebnis wahr einstellt, d.h. '1'B. Nunmehr ist der ursprüngliche Ausdruck auf den wesentlich einfacheren Ausdruck

```
'1'B & '1'B
```

zurückgeführt. Die logische Und-Verknüpfung führt schließlich auf das Ergebnis '1'B, ist somit ebenfalls wahr. Dadurch kommt es zur Ausführung der THEN-Klausel des IF-Statements. Die mit MB beginnende Schleife wird infolgedessen solange durchlaufen, bis CC den Wert -521 angenommen hat. Erst dann kommt es zur Ausführung der ELSE-Klausel.

Folgen in einem Ausdruck verschiedenartige Operationen aufeinander und wird durch Klammern die Reihenfolge der Operationen nicht eindeutig festgelegt, so werden die Operationen in einer bestimmten Reihenfolge ausgeführt, die durch die der Operatoren bestimmt ist. Es gilt folgende Rangordnung für die Operatoren (Abb. 12.7.).

	Stufe	Operation(en)
höchste	1	¬, ** ,
	2	* , /
	3	+ , -
	4	| |
	5	>= , > , ¬> , ¬= , < , ¬< , <= , =
	6	&
niedrigste	7	|

Abb. 12.7. Rangordnung der Operatoren

Enthält ein Ausdruck Operatoren verschiedener Stufen, so werden Operationen höherer Rangordnungen vor den Operationen niedrigerer Rangordnungen ausgeführt. Bei Ausdrücken mit mehreren Operationen auf gleicher Stufe werden diese in der Reihenfolge von links nach rechts ausgeführt. Von dieser Regel ausgenommen ist jedoch die erste

Stufe, bei der die Operationen von rechts nach links ablaufen. Bei allen Operationen werden im Bedarfsfall vor ihrer Ausführung Datenumwandlungen vorgenommen.

Aufgabengruppe 29

1) Schreibe ein Programm, das einen x-Wert von einer Lochkarte liest und die Werte von y und z druckt. Der x-Wert ist als natürliche Zahl vorzugeben. Für z sollen der Reihe nach die Werte $\emptyset, 1, 2, \cdots, x$ eingesetzt werden. Zur Berechnung von y diene die Beziehung

$$y = \frac{4z}{3+z-2x}$$

Sollte der Nenner den Wert $\emptyset$ annehmen, so ist die damit auftretende Bedingung durch die Anweisung

ON ZERODIVIDE ···

abzufangen und y in diesem Fall der Wert $\emptyset$ zuzuweisen.

2) Ändere die Aufgabe 5) der Aufgabengruppe 28 wie folgt ab:

Nachdem die letzte Karte eingelesen worden ist, ist nach einem doppelten Zeilentransport die Summe aller x-Werte zu drucken.

3) Schreibe ein PL/1-Programm zur Berechnung von

$$\sum_{n=2}^{\infty} (-1)^n \cdot \frac{1}{\ln(n)}$$

Es sind solange Glieder zu addieren, bis ein Glied gebildet ist, das absolut kleiner als $\emptyset.\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset 1$ ist.

4) Schreibe das Programm von Aufgabe 3) so um, daß es möglich ist, solange Glieder hinzuzufügen, bis das nächste Glied absolut kleiner als

a) $\emptyset,\emptyset 1$, b) $\emptyset,\emptyset\emptyset 1$, c) $\emptyset,\emptyset\emptyset\emptyset 1$, d) $\emptyset,\emptyset\emptyset\emptyset\emptyset 1$, e) $\emptyset,\emptyset\emptyset\emptyset\emptyset\emptyset 1$ ist!

Schließlich ist in das Programm eine Schleife so einzubauen, daß es möglich ist, während eines Programmablaufes alle fünf Fehlergrenzen zu berücksichtigen.

5) Es soll der Maximalfehler abgeschätzt werden, der sich ergibt, wenn

$$\sqrt{\frac{1+x}{1+y}}$$

durch

$$1+\frac{x-y}{2}$$

angenähert wird. Es soll angenommen werden, daß sowohl die x-Werte als auch die Y-Werte zwischen $-\emptyset.1$ und $\emptyset.1$ liegen. Schreibe deshalb ein Programm, in dem die Werte von

$$f(x,y) = \sqrt{\frac{1+x}{1+y}} - 1 - \frac{x-y}{2}$$

im Intervall von $x=-\emptyset.1$ bis $x=\emptyset.1$ mit einer Schrittweite von $x=\emptyset.\emptyset 1$ berechnet werden. Das gleiche Intervall und die gleiche Schrittweite ist für y anzusetzen. Es ist nur der maximale Fehler auszugeben.

6) Das Programm zu Aufgabe 5) ist so zu modifizieren, daß eine Tabelle ausgegeben wird, die den maximalen Fehler für alle 1Ø1 verschiedenen x-Werte enthält. Zu jedem dieser Fehler soll der dazugehörige x-Wert ausgegeben werden.

7) `DECLARE B(3,3) BIT(2) ;`
Durch diese Erklärung wird ein Quadrat beschrieben, das in neun gleichgroße kleinere Quadrate unterteilt ist. Ein Spieler soll nun gegen den Computer um die Besetzung der einzelnen Quadrate spielen. Ein Wert von '1Ø'B bedeutet, daß der Spieler das entsprechende Quadrat besetzt hat, ein Wert von 'Ø1'B, daß der Computer das entsprechende Quadrat besetzt hat. Durch 'ØØ'B wird angezeigt, daß das betreffende Quadrat leer ist, der Wert von '11'B ist bedeutungslos.

Der Spieler soll die erste Bewegung ausführen können. Durch die Eingabe von zwei ganzen Zahlen (Zeile und Spalte des großen Quadrates) soll er festlegen können, welches kleine Quadrat er zu besetzen wünscht. Das Programm soll anschließend zwischen den besetzten und den freien Quadraten unterscheiden können und daraus resultierend ein freies Quadrat auswählen und dem Computer zuweisen können. Nach jedem dieser Spielschritte soll die Besetzung der einzelnen Quadrate ausgedruckt werden, wobei für den Spieler ein X und für den Computer ein O verwendet werden soll. Danach soll der Vorgang wiederholt werden können.

Zu einem Spielabbruch soll es dann kommen, wenn es entweder dem Spieler oder dem Computer gelingt, drei in einer Zeile oder Spalte gelegene Quadrate zu besetzen. Es ist nicht erforderlich, daß das Programm so aufgebaut ist, daß für den Computer die günstigsten Quadrate ausgewählt werden.

8) Durch ein Programm soll ein Lochkartenpaket eingelesen werden. Auf der ersten Karte ist der Wert von n, auf den nächstfolgenden Karten ist jeweils ein x-Wert abgelocht. Für jeden eingelesenen x-Wert sind folgende Berechnungen auszuführen:

I. $y = 3x^2 + 5x + 4$

II. $z = 2x^2 + 6x + 3$

III. $w = \begin{cases} \frac{y}{z} & \text{für } z \neq Ø \\ 17,4 & \text{für } z \neq Ø \end{cases}$

a) Ist n=1, so sind die Werte von x, y und z zu drucken.

b) Ist n=2, so sind die Werte von x, y, z und w zu drucken.

c) Ist n=3, so sind die Werte von x, y und w zu drucken, wobei ein Zwischenraum zwischen den Werten von y und w bleiben soll, der so groß ist wie in den anderen Fällen der für den z-Wert vorgesehene Platz.

d) Ist n=4, so sind die Werte von x und w zu drucken, wobei auch hier wieder Zwischenräume verbleiben sollen, die Platz für die y- und z-Werte bieten.

Um diese Programmaufgabe erfüllen zu können, sollen vier verschiedene PUT-Anweisungen codiert werden. Jeder PUT-Anweisung soll eine GOTO-Anweisung folgen. Das den Fällen a) bis d) zugeordnete PUT-Statement soll mittels einer Markenbereichsvariablen gefunden werden, wobei der Wert von n als Index verwendet wird. Bei der Eingabe ist außerdem zu prüfen, daß n nur die Werte 1, 2, 3 oder 4 annimmt.

12.9. Statementgruppen

Nehmen wir an, daß wir ein Programm schreiben sollen, das zur Bestimmung der Lösungen quadratischer Gleichungen in der Normalform

$$ax^2 + bx + c = \emptyset$$

dienen soll. Zur Eingabe gelangen Lochkarten, auf denen die Werte der Koeffizienten a, b und c abgelocht sind. Zur Ausgabe über den Drucker sollen die Werte von a, b und c gelangen sowie die beiden Lösungen

$$x_1 + iy_1$$
$$x_2 + iy_2$$

Wir setzen weiterhin voraus, daß für a, b und c nur reelle Werte eingegeben werden. Die Lösungen können dagegen sowohl reell als auch komplex sein. Ein Programm, das dieser Aufgabenstellung gerecht wird, könnte den in der Abb. 12.8. dargestellten Aufbau besitzen.

```
/* BESTIMMUNG DER LOESUNGEN QUADRATISCHER GLEICHUNGEN */
QUAGL:  PROCEDURE OPTIONS(MAIN) ;
          ON ENDFILE(SYSIN) GOTO ENDE ;
   P1:    GET LIST (A,B,C) ;
          D = B*B - 4*A*C ;
          E = -B/(A+A) ;
          IF  D<Ø  THEN GOTO NEG ;
                   ELSE GOTO POS ;
  POS:    Y1,Y2 = Ø ;
          R = SQRT(D)/(A+A) ;
          X1 = E + R ;
          X2 = E - R ;
          GOTO DRU ;
  NEG:    X1,X2 = E ;
          Y1 = SQRT(-D)/(A+A) ;
          Y2 = - Y1 ;
  DRU:    PUT LIST (A,B,C,X1,Y1,X2,Y2) ;
          GOTO P1 ;
 ENDE:    ;
        END QUAGL ;
```

Abb. 12.8. Programm zur Lösung quadratischer Gleichungen

Betrachtet man dieses Programm kritisch, so würde man es als sehr vorteilhaft empfinden, wenn man in die THEN- und ELSE-Klauseln des IF-Statements eine ganze Gruppe von Statements schreiben könnte und nicht gerade nur eines, wie wir es bisher kennengeiernt haben. Eine Möglichkeit, diese Beschränkung zu umgehen, besteht darin, eine DO-Gruppe zu verwenden. Ihre Form lautet:

```
marke:  DO ;
        ein oder mehrere statements
        END  marke  ;
```

Die Angabe einer Marke ist freigestellt.

Stößt man auf ein DO-Statement, so bilden alle Statements bis zum abschließenden END-Statement eine Gruppe. Mit Ausnahme der ON-Statements dürfen Gruppen überall dort eingesetzt werden, wo auch ein einzelnes Statement verwendet werden kann. Weiterhin

dürfen DO-Gruppen in andere DO-Gruppen eingebettet werden. DO-Gruppen sind sehr zweckmäßig in THEN- und ELSE-Klauseln von IF-Statements. Das kann sehr schön am Beispiel zur Bestimmung der Lösungen quadratischer Gleichungen gezeigt werden. Bei Verwendung von Gruppen kann das in Abb. 12.8. dargestellte Programm so umgeschrieben werden, wie es die Abb. 12.9. zeigt.

```
/* BESTIMMUNG DER LOESUNGEN QUADRATISCHER GLEICHUNGEN */
QUAGL:  PROCEDURE OPTIONS(MAIN) ;
          ON ENDFILE(SYSIN)  GOTO ENDE ;
   P1:    GET LIST (A,B,C) ;
          D = B*B - 4*A*C ;
          E = -B/(A+A) ;
          IF  D<Ø  THEN DO ;
                        X1,X2 = E ;
                        Y1 = SQRT(-D)/(A+A) ;
                        Y2 = - Y1 ;
                        END ;
                   ELSE DO ;
                        R = SQRT(D)/(A+A) ;
                        X1 = E + R ;
                        X2 = E - R ;
                        Y1,Y2 = Ø ;
                        END ;
          PUT LIST (A,B,C,X1,Y1,X2,Y2) ;
ENDE:     ;
        END QUAGL ;
```

Abb. 12.9. Programm zur Lösung quadratischer Gleichungen mit DO-Gruppen (Änderung des Programmes von Abb. 12.8.)

Es fällt besonders auf, daß es uns gelungen ist, nicht weniger als drei Marken zu eliminieren. Im allgemeinen ist um so leichter, ein Programm auf Irrtümer zu durchsuchen, je weniger Marken es enthält.

Wir wollen nunmehr einen Fall betrachten, der sich dadurch auszeichnet, daß zwei DO-Gruppen ineinander geschachtelt sind.

```
A:  PROCEDURE ;
      .
      .
B:  DO ;
      .
      .
C:  DO ;
      .
      .
    END C ;
    END B ;
    END A ;
```

Die letzten drei Statements hätten wir ebenso gut auch ohne Marke schreiben können, also

```
END ;   END ;   END ;
```

Treten wie hier mehrere END-Statements unmittelbar hintereinander auf, so ist es gestattet, nur ein einziges END-Statement zu schreiben, doch muß sich dieses dann auf eine Marke beziehen, und zwar auf diejenige, die das DO-Statement der äußersten Gruppe bzw. Prozedur kennzeichnet. In unserem Fall müßten wir also codieren:

```
END A ;
```

Durch dieses Statement wird erkannt, daß alle Zusammenfassungen beendet sind, die beim Statement beginnen, das durch die Marke A gekennzeichnet ist. Die Verwendung eines einzelnen END-Statements macht die Programme sicher nicht einfacher lesbar; infolgedessen sollte der Anfänger besonders sorgfältig mit dieser Möglichkeit des END-Statements umgehen.

Das DO-Statement kann genau so wie die anderen Statements durch mehr als eine Marke gekennzeichnet werden.

12.10. DO-Statements zur Einleitung von Schleifen

Wir wollen einen Programmabschnitt betrachten, der zur Berechnung von Kubikwurzeln dient. Zunächst sei das Rechenverfahren vorgestellt, das diesem Abschnitt zugrunde liegt. Es soll die dritte Wurzel aus n bestimmt werden. Bezeichnen wir sie mit x, so können wir als ersten Nährungswert für diese Wurzel den Wert

$$x_0 = 1$$

annehmen und den nächsten Näherungswert nach der Formel

$$x_1 = (2x_0 + n/x_0^2)/3$$

berechnen. Nach diesem Schritt setzen wir $x_0=x_1$ und berechnen nach der obigen Formel einen neuen Näherungswert x_1. Diesen Prozeß setzen wir solange fort, bis wir auf zwei aufeinanderfolgende Näherungswurzeln stoßen, die innerhalb der vorgegebenen Fehlertoleranz gleich sind. Dieses Iterationsverfahren [1] wollen wir numerisch einmal verfolgen, indem wir die Kubikwurzeln aus 4 berechnen (Abb. 12.10.).

	n=4
	$x_0=1$
1. Schritt:	$x_0=(2\cdot 1+4/1)/3=2$
2. Schritt:	$x_0=(2\cdot 2+4/4)/3=1{,}667$
3. Schritt:	$x_0=(3{,}333+4/2{,}778)/3=1{,}591$
4. Schritt:	...
⋮	⋮

Abb. 12.10. Iterationsschritte zur Berechnung von $\sqrt[3]{4}$

Je öfter wir die Rechnung wiederholen, um so mehr nähern wir uns dem Wert 1,5874..., dem Wert also, dessen erste fünf Ziffern nicht mehr verbessert werden können, wenn wir $\sqrt[3]{4}$ bestimmen.

Es folgt nunmehr der Programmabschnitt in der PL/1-Sprache, der zur Berechnung von Kubikwurzeln dienen kann (Abb. 12.11.).

1) Iteration → Wiederholung.

```
        XALT = Ø ;    XNEU = 1 ;
M1:     IF  ABS(XNEU-XALT)<Ø.ØØØ1  THEN GOTO FERTIG ;
        XALT = XNEU ;
        XNEU = (2*XALT + N/XALT**2) / 3 ;
        GOTO M1 ;

FERTIG: ...
```

Abb. 12.11. Programmabschnitt zur Berechnung von Kubikwurzeln

Da es sehr häufig vorkommt, daß derartige Rechnungswiederholungen zu codieren sind, wurde ein besonderes PL/1-Statement geschaffen, um derartige Verfahren möglichst einfach codieren zu können. Seine allgemeine Form lautet (im Zusammenhang mit anderen Statements gesehen):

```
DO WHILE(ausdruck) ;
  statement(s)
END ;
```

Die in diese DO-Gruppe aufgenommenen Statements werden solange wiederholt, solange der in Klammern stehende Ausdruck wahr ist. Sobald der Ausdruck falsch wird, werden die Statements in der Gruppe übergangen und die Ablaufsteuerung des Programmes verzweigt zu dem Statement, das auf das END-Statement dieser Gruppe folgt. Ist der Ausdruck bereits falsch, wenn der Programmablauf zum ersten Mal auf das DO-Statement stößt, so werden die in der Gruppe stehenden Statements überhaupt nicht ausgeführt. Für unser Beispiel zur Berechnung von Kubikwurzeln können wir infolgedessen die Statementfolge von Abb. 12.12. schreiben.

```
XALT = Ø ;    XNEU = 1 ;
DO WHILE(ABS(XNEU-XALT)>Ø.ØØØ1) ;
   XALT = XNEU ;
   XNEU = (2*XALT + N/XALT**2) / 3 ;
END ;
```

Abb. 12.12. Iterative Schleife zur Berechnung von Kubikwurzeln unter Benutzung der DO-Anweisung

Wollen wir beispielsweise die Summe der Kuben der Elemente eines Bereiches bilden, so wenden wir zweckmäßig eine andere Form des DO-Statements an. Setzen wir voraus, daß BER ein Bereich (Vektor) von 19 Elementen ist und daß wir die Summe der Kuben aller Bereichselemente bilden wollen. Wir können den entsprechenden Programmabschnitt so aufbauen, wie es die Abb. 12.13. zeigt.

```
        DCL  BER(19) DEC FIXED(4,Ø),
             KUB_SUM DEC FIXED(13,Ø) INIT(Ø) ;

        J = 1 ;
MA:     IF  (J>19)  THEN GOTO WEIT ;
        KUB_SUM = KUB_SUM + BER(J)**3 ;
        J = J + 1 ;
        GOTO MA ;
WEIT:   ...
```

Abb. 12.13. Programmabschnitt zur Berechnung der Summe der Kubikzahlen für die Elemente eines Vektors

Würden wir eine DO-Gruppe benutzen wollen, so hätten wir nach den bisherigen Erkenntnissen codieren können:

```
DO  WHILE(J<=19) ;
    :
    :
    J = J + 1 ;
END ;
```

Wir können jedoch eine andere Form der DO-Anweisung viel zweckmäßiger einsetzen:

```
DO  J=1 TO 19 BY 1 ;
    KUB_SUM = KUB_SUM + BER(J)**3 ;
END ;
```

Nach dieser ersten Vorstellung der DO-Anweisung in einer anderen Form wollen wir uns jetzt mit der allgemeinen Form der DO-Anweisung beschäftigen. Sie lautet:

DO $\underline{v}=\underline{a_1}$ TO $\underline{a_2}$ BY $\underline{a_3}$ WHILE($\underline{a_4}$) ···

Hierbei bedeutet $\underline{v}$ eine skalare Variable (jedoch keine Bereichsvariable), die indiziert sein kann und vom arithmetischen Typ oder Kettentyp sein darf, ja sogar das Attribut LABEL besitzen kann ($\underline{v}$ wird oft als Laufvariable bezeichnet); alle $\underline{a_i}$ mit i=1,2,3,4 stellen Ausdrücke dar. Die beiden Schlüsselwörter TO und BY dürfen zusammen mit den ihnen beigeordneten Ausdrücken $\underline{a_2}$ bzw. $\underline{a_3}$ in beliebiger Reihenfolge aufgeführt werden. Somit sind z.B. die beiden Anweisungen

```
DO  J=1 TO 19 BY 1 ;
```

und

```
DO  J=1 BY 1 TO 19 ;
```

einander gleichwertig.

Unser oben eingeführtes allgemeines DO-Statement wird durch die Kompilierer in eine Instruktionsfolge aufgelöst, die der in der Abb. 12.14. dargestellten Folge von PL/1-Statements entspricht.

```
marke1:  IF  (a3>=Ø)&(v>a2)   (a3<Ø)&(v<a2)  THEN GOTO WEIT ;
marke3:  IF  (a4)  THEN ;   ELSE GOTO WEIT ;
         statement1
         statement2
         :
         statementn
marke2:  v = v + a3 ;
         GOTO marke1 ;
  WEIT:  ···
```

Abb. 12.14. Auflösung der DO-Anweisung

Fehlt WHILE($\underline{a_4}$) im DO-Statement, so wird bei der Auflösung selbstverständlich

```
marke3:  IF  (a4)  THEN ;   ELSE GOTO WEIT ;
```

weggelassen.

Fehlt TO $\underline{a_2}$, dann wird das Statement mit der Marke $\underline{marke_1}$ durch die Leeranweisung ersetzt. Fehlt dagegen BY $\underline{a_3}$, dann wird $\underline{a_3}$ standardmäßig durch 1 ersetzt.

Der rechts vom Ergibtzeichen stehende Teil eines DO-Statements wird als Spezifikation bezeichnet. Ein einziges DO-Statement kann beliebig viele verschiedene Spezifikationen enthalten. Die Spezifikationen sind voneinander durch Kommas zu trennen. Somit bedeutet

```
DO  J=1 TO 19,23,25 TO 37 BY 2 WHILE(DY>1.ØE-4) ;
```

z.B., daß die in der Gruppe liegenden Anweisungen für J=1,2,···,19,23,25,27,29,31, 33,35 und 37 ausgeführt werden, falls bei den Werten von 25 an aufwärts nicht inzwischen der Wert der Variablen DY=0,0001 geworden ist.

Bei diesem Beispiel beachte man insbesonders die Spezifikationen 19 und 23. Werden nämlich TO und BY weggelassen, so wird die zur Schleife gehörende Statementgruppe nur einmal mit dem angegebenen Wert durchlaufen.

Die DO-Anweisung ist eine der mächtigsten Anweisungen der Sprache. Im Interesse eines optimalen Programmablaufes sollten wir die in der DO-Anweisung angegebene Laufvariable möglichst als ganzzahlige duale Variable in der Festpunktdarstellung wählen. Grundsätzlich, das sei noch einmal betont, bestehen jedoch hinsichtlich des Typs der Laufvariablen nur geringfügige Einschränkungen. Die Laufvariable wird hin und wieder auch als DO-Index bezeichnet.

12.11. Indizierungen in Datenlisten bei Anweisungen der reihenweisen EA

Eine besondere Form der Spezifikationen, die wir bei DO-Anweisungen kennengelernt haben, ist in den Datenlisten von GET- und PUT-Anweisungen verfügbar.

Innerhalb dieser Listen kann nämlich das Schlüsselwort DO gebraucht werden, dem in üblicher Weise die Variable v und die Spezifikationen folgen können, so wie sie beim DO-Statement ausführlich besprochen wurden. Das Schlüsselwort END darf hier jedoch keineswegs verwendet werden. Die allgemeine Form unter Berücksichtigung mehrerer Spezifikationen lautet also:

```
(de [,de]···  DO v=s [,s]···)
```

Hierbei bedeuten:

de	Datenelement
v	Laufvariable
s	Spezifikation

Man beachte, daß die gesamte Liste inklusive der DO-Option in Klammern zu stehen hat.

Die DO-Optionen können außerdem ineinander geschachtelt sein. Bei allen Verschachtelungen müssen jedoch Klammern gebraucht werden; nur dadurch ist die Eindeutigkeit der Zuordnungen der DO-Optionen gewährleistet.

Zur ersten Illustration wollen wir das Statement

```
GET LIST (A, (B(I),C(I+1), (D(I+J) DO J=1 TO 2) DO I=2 TO 5), E)
```

betrachten. Dieses Statement ist äquivalent der Statementfolge:

```
GET LIST(A) ;
DO  I=2 TO 5 ;
    GET LIST(B(I),C(I+1)) ;
    DO  J=1 TO 2 ;
        GET LIST(D(I,J)) ;
    END ;
END ;
GET LIST(E) ;
```

Vom Datenträger werden die Werte in der folgenden Reihenfolge gelesen und den Variablen bzw. Bereichselementen nach evtl. erforderlichen Umwandlungen zugewiesen.

```
A ——— B(2) ——— C(3) ——— D(2,1) ——— D(2,2) ———
      B(3) ——— C(4) ——— D(3,1) ——— D(3,2) ———
      B(4) ——— C(5) ——— D(4,1) ——— D(4,2) ———
      B(5) ——— C(6) ——— D(5,1) ——— D(5,2) ——— E
```

Die Verwendung von Wiederholungsspezifikationen in den Datenlisten von EA-Anweisungen bietet uns zwei Vorteile. In vielen Fällen können wir uns das Schreiben umfangreicher Datenlisten ersparen (im obigen Beispiel hätten wir sonst alle 18 Elemente in die Liste aufnehmen müssen) oder aber das Schreiben komplizierter DO-Gruppen. In anderen Fällen können wir mit ihrer Hilfe Probleme elegant lösen, die wir auf andere Weise nur schwer beherrschen würden. Wie meinen damit z.B., daß wir mit variablen Bereichsumfängen jonglieren können. Angenommen, wir haben ein Programm, das Matrizen von verschiedenen Ordnungen drucken soll. Der Programmierer kennt dabei die Ordnung der Matrizen nicht, erst beim Ablauf des Objektprogrammes ist diese bekannt. Bewältigt werden kann dieses Problem, indem man für die Variable in der DO-Gruppe veränderliche Grenzen angibt (TO···). Als Beispiel hierfür soll ein Statement niedergeschrieben werden, mit dem quadratische Matrizen n-ter Ordnung gedruckt werden können.

```
PUT LIST ((A(I,J) DO J=1 TO N) DO I=1 TO N)) ;
```

Wir können uns nach Auflösung dieses Statements in zwei ineinander geschachtelte Schleifen leicht davon überzeugen, daß in diesem Fall die Matrixelemente zeilenweise gedruckt werden. Erinnern wir uns: Wenn A als Bereich mit zwei Dimensionen jeweils in den Grenzen 1 und N erklärt ist, dann ist auch die Anweisung

```
PUT LIST (A) ;
```

akzeptabel, um die gleiche Ausgabe zu erzielen. Variiert jedoch während der Programmausführungszeit der Bereichsumfang, so müssen die genannten Spezifikationen doch verwendet werden.

Doch kehren wir zum ursprünglichen Statement zurück. Wir sollten einmal die Funktion der drei aufeinanderfolgenden geöffneten Klammern im Anschluß an das Schlüsselwort LIST beobachten und sie in Verbindung mit den sie zu einem Klammernpaar ergänzenden schließenden Klammern bringen. Das äußerste Klammernpaar ist erforderlich, weil Datenlisten nun einmal von Klammern begrenzt werden müssen. Das nächste Klammernpaar umgibt die von der Laufvariablen I abhängige Wiederholungsspezifikation. Das innerste Klammernpaar schließlich ist notwendig, um die von der Laufvariablen J abhängige Wiederholungsspezifikation einzuschließen.

Werden mehrere Wiederholungsspezifikationen für eine Laufvariable gebraucht, so sind diese wie gewöhnlich durch Kommas voneinander zu trennen; auf diese Weise entsteht die bereits im Abschnitt 12.10. vorgestellte Spezifikationsliste. Die zu einer Laufvariablen gehörenden Spezifikationen werden also nicht anders behandelt wie die Datenelemente in Datenlisten. Und in der Konsequenz, sollen mehrere Datenelemente von einer Laufvariablen EA-mäßig gesteuert werden, so sind diese ganz normal durch Kommas voneinander zu trennen. Nach einer kurzen Exkursion über bereits bekannte Fakten über Formatlisten wollen wir hierzu ein Beispiel bringen.

An dieser Stelle sollten wir uns zunächst etwas näher mit den Formatlisten bei der formatgesteuerten Eingabe/Ausgabe befassen. Normalerweise besteht eine Eins-zu-Eins-Beziehung zwischen den Datenelementen in der Datenliste und den Formatelementen in der Format-

liste, sieht man einmal von den Steuerungsformatelementen ab (PAGE, SKIP, X usw.). Ist jedoch die Anzahl der auszugebenden Datenelemente nicht bekannt oder veränderlich, so ist die Angabe einer richtigen Anzahl von Formatelementen nicht möglich. Zur Lösung derartiger Probleme sind bekanntlich folgende Regeln festgelegt worden (siehe Kap. 10.).

a) Wenn die Datenelemente der Datenliste erschöpft sind, so gilt die Ausgabe als beendet.

b) Wenn die Formatelemente (abgesehen von den Steuerungsformatelementen) der Formatliste erschöpft sind, so werden die Formatelemente vom Beginn der Formatliste an wiederholt.

Gleichzeitig zur Untermauerung dieser altbekannten Regeln wollen wir nun das bereits angekündigte Beispiel bringen:

```
PUT EDIT (((H(I),G(I,J) DO J=1 TO 4) DO I=1 TO 9))
         (PAGE,(9)(SKIP,E(2Ø,8),X(6),(4)E(2Ø,8))) ;
```

Durch diese Druckanweisung werden neun Zeilen mit je 5 Werten auf eine neue Seite gedruckt.

Die beiden in diesem Fall benutzten Wiederholungsfaktoren brauchen nicht unbedingt vorzeichenfreie ganzzahlige Dezimalkonstanten zu sein. Sie können vielmehr als beliebige Ausdrücke niedergeschrieben werden, vorausgesetzt, sie sind in Klammern eingeschlossen. Das obige Beispiel könnte deshalb auch die folgende Gestalt annehmen:

```
PUT EDIT (((H(I),G(I,J) DO J=1 TO LIMIT_J) DO I=1 TO LIMIT_I))
         (PAGE,(LIMIT_I)(SKIP,E(2Ø,8),X(6),(LIMIT_J)E(2Ø,8))) ;
```

Aufgabengruppe 30

1) Das in Aufg. 3) der Aufgabengruppe 29 vorgestellte Problem soll unter Verwendung von DO-Gruppen neu programmiert werden (Verwendung der WHILE-Option).

2) Das in Aufg. 6) der Aufgabengruppe 29 vorgestellte Problem soll unter Verwendung des DO-Statements neu programmiert werden.

3) Das in Aufg. 1) der Aufgabengruppe 29 vorgestellte Problem soll unter Verwendung des DO-Statements neu programmiert werden.

4) Unter Verwendung einer DO-Gruppe soll der Wert von

$$\sum_{n=1}^{\infty} (-1)^n \frac{3^n + 4^n}{2 \cdot 6^n}$$

berechnet werden.

Die Berechnung der Glieder ist so lange fortzusetzen, bis ein Glied gefunden ist, daß absolut kleiner als Ø.ØØØØ1 ist.

5) Eine Matrix A weist vier Zeilen und drei Spalten auf. Ihre Elemente sind ganze Dezimalzahlen. Es ist ein Programm zu schreiben, das die Matrixelemente einliest, die auf Karten wie folgt abgelocht sind:

Spalte 2:	Zeilenindex des Elementes (Werte 1 bis 4)
Spalte 4:	Spaltenindex des Elementes (Werte 1 bis 3)
Spalten 5 bis 11:	Matrixelement

Nach dem Einlesen aller Matrixelemente ist die Matrix vierzeilig und spaltengerecht zu drucken. Unterhalb der Matrix ist außerdem die Nachricht

```
ANZAHL DER NULLELEMENTE = xx
```

zu drucken. Für xx ist die Anzahl der Matrixelemente zu setzen, die gleich Null sind. Ein Matrixelement, für das kein Wert eingelesen wird, ist gleich Null zu setzen.

6) Ändere das Programm von Aufg. 5) in zweierlei Hinsicht ab:

   a) Nach dem Einlesen der Werte für die Matrixelemente ist die Anzahl der Matrixelemente zu drucken, für die kein Wert eingelesen wurde.

   b) Es ist die Möglichkeit zuzulassen, daß zwei oder mehr Werte für das gleiche Matrixelement eingelesen werden. In diesem Fall soll der erste eingelesene Wert beibehalten, alle übrigen Werte zurückgewiesen werden.

7) Schreibe ein Programm, das die Werte der Funktion

$$f(x) = \sin x + \cos(2x) + 3.5 \sin(3x)$$

berechnet und in einer Tabelle speichert. Die x-Werte sollen dabei das Intervall von Ø bis Ø.5 mit einer Schrittweite von Ø.Ø1 durchlaufen.

8) Ergänze das Programm zu Aufgabe 7) so, daß mit ihm Karten gelesen werden können, auf denen ein f(x)-Wert abgelocht ist, der nach Voraussetzung kleiner als 3.4 sein soll. Durchsuche die Tabelle Schritt für Schritt, um den dazugehörigen x-Wert zu bestimmen bzw. die zwei Vielfachen von Ø.Ø1 zu finden, zwischen denen dieser x-Wert liegt. Der ermittelte x-Wert ist zu drucken; im letzteren Fall soll derjenige ausgegeben werden, dessen Funktionswert die kleinere absolute Differenz zum eingelesenen Funktionswert aufweist. Durch diese Aufgabe wird eine einfache (lineare) Methode des Tabellenlesens vorgestellt, die jedoch nicht gerade als elegant bezeichnet werden kann.

12.12. Strukturen

Angenommen, vor uns steht die Aufgabe, eine Zeichenkette zu definieren, die ein Datum aufnehmen soll. Wir könnten schreiben

```
DCL   TAGE CHAR(2),
      MONATE CHAR(2),
      JAHRE CHAR(2) ;
```

Verfahren wir so, so besteht für uns keine Möglichkeit, daß wir uns auf das Datum als Ganzes beziehen.

Jeder in einem DECLARE-Statement aufgeführten Variablen können wir aber eine sogenannte Stufennummer beifügen. Die Stufennummer muß als ganzzahlige dezimale Konstante geschrieben werden (beim System IBM /370 muß diese kleiner als 256 sein). Unter Benutzung derartiger Stufennummern können wir unser obiges Beispiel wie folgt umschreiben:

```
DECLARE   1 DATUM,
            2 TAGE CHAR(2),
            2 MONATE CHAR(2),
            2 JAHRE CHAR(2) ;
```

Wenn wir uns jetzt des Namens DATUM bedienen, so schließen wir damit die drei Elemente ein, aus denen sich diese Variable zusammensetzt. Eine derartige untergliederte Anordnung von Daten bezeichnen wir als *Struktur*. Eine Struktur kann skalare Variablen, Bereichsvariablen und andere Strukturen enthalten. Ist eine Struktur durch die Stufennummer 1 gekennzeichnet, so wollen wir eine solche Struktur als *Hauptstruktur* bezeichnen, alle anderen Strukturen, die notwendigerweise in

Hauptstrukturen enthalten sein müssen, als Unterstrukturen. Eine nicht weiter untergliederte Strukturstufe heißt Strukturelement.

Wird in einem DECLARE-Statement eine Variable ohne Stufennummer geschrieben, so wird ihr standardmäßig die Stufennummer 1 zugeordnet.

Wir wollen nunmehr die folgende Struktur näher betrachten, um uns mit den eingeführten Bezeichnungen vertraut zu machen:

```
DECLARE  1 A,
           2 B CHARACTER(16),
           2 C,
             7 D(4) FIXED(5),
           2 E(3),
             5 F CHARACTER(8),
             5 G(4) BIT(4) ;
```

Wir erkennen, daß die Hauptstruktur A in die Unterstrukturen B, C und E untergliedert ist.

Der einzige Bestandteil von C ist der Bereich (Vektor) D, welcher vier Elemente besitzt. Die Unterstruktur E ist als Bereich von drei Elementen organisiert. Jedes Bereichselement wiederum zerfällt in zwei Strukturelemente, in die skalare Variable F und in den Bereich G. Beziehen wir uns auf die Bereichselemente von G, so müssen wir zwei Indizes angeben, einmal den Index, der der Unterstruktur E zugeordnet ist, und zum anderen den Index des Bereiches G selbst. Durch diese Strukturerklärung (Hauptstruktur ist A, Unterstrukturen sind B, C und E, Strukturelemente sind B, D, F und G) wird die Anordnung von Elementen erzeugt, die in der Abb. 12.15. dargestellt ist.

B	Strukturelement B
D(1) D(2) D(3) D(4)	Unterstruktur C mit dem Strukturelement D
F(1) G(1,1) G(1,2) G(1,3) G(1,4)	1. Bereichselement der Unterstruktur E
F(2) G(2,1) G(2,2) G(2,3) G(2,4)	Unterstruktur E mit den Strukturelementen F und G
F(3) G(3,1) G(3,2) G(3,3) G(3,4)	3. Bereichselement der Unterstruktur E

Abb. 12.15. Beispiel für eine Struktur

Genau so wie Bereiche Strukturelemente sein können (hier: G), so können sich die Bereichselemente in Strukturen untergliedern, wie es hier bei E der Fall ist (die Bereichselemente setzen sich hier aus einer skalaren Variablen F und einem Bereich G zusammen).

Strukturnamen können in Ergibtanweisungen verwendet werden. Die Strukturvariable(n) auf der linken Seite des Ergibtzeichens müssen dabei dieselbe Gliederung aufweisen wie die Strukturvariablen auf der rechten Seite. Es sei ausdrücklich betont, daß nur die Strukturgliederungen übereinstimmen müssen, die Stufennummern selbst dürfen durchaus verschieden sein. Ein Beispiel hierfür folgt in Abb. 12.16.

```
DCL   1 A,
        2 B CHAR(16),
        2 C,
          3 D(2) CHAR(4),
      1 H,
        2 K CHAR(16) INIT('MARIANNE'),
        2 R,
          8 U(2) CHAR(4) INIT((2)(4)'Ø') ;
A = H;
```

Abb. 12.16. Beispiel für eine Strukturergibtanweisung

Strukturergibtanweisungen werden durch den Kompilierer in eine Reihe von skalaren Ergibtanweisungen aufgelöst. So wird

```
A = H ;
```

zunächst in die beiden Ergibtanweisungen

```
B = K ;
C = R ;
```

übergeführt. Die letzte schließlich wird in die zwei Ergibtanweisungen

```
D(1) = U(1) ;
D(2) = U(2) ;
```

zerlegt.

Auf der rechten Seite einer solchen Ergibtanweisung darf selbstverständlich ein Ausdruck erscheinen. Wie dieser aber auch aussieht, alle Operationen werden grundsätzlich zwischen den an gleicher Stelle der Gliederung stehenden, einander entsprechenden Strukturelementen ausgeführt, d.h. also, als ob eine Reihe von einzelnen Ergibtanweisungen niedergeschrieben worden wäre.

In Strukturergibtanweisungen darf außerdem die Option BY NAME verwendet werden. In einem solchen Fall ist es nicht notwendig, daß die auf den linken bzw. rechten Seiten des Ergibtzeichens stehenden Strukturen die gleiche Gliederung aufweisen, doch muß mindestens ein Strukturelement jeder Struktur den gleichen Namen haben. Diese Option sei an einem Beispiel erläutert (Abb. 12.17.).

```
DCL   1 J,
        2 K CHAR(4),
        2 L CHAR(4) ;
DCL   1 M,
        2 L CHAR(4),
        2 P CHAR(4) ;
J = M, BY NAME ;
```

Abb. 12.17. Beispiel für die Option BY NAME in Strukturergibtanweisungen

Die BY NAME - Option bewirkt, daß Ergibtanweisungen zwischen Strukturelementen gleichen Namens, unabhängig von ihrer Stellung innerhalb der Strukturen, erzeugt werden sollen. Im obigen Beispiel kommt es also zu der folgenden Zuweisung:

```
L → in J  =  L → in M
```

Sind wir gezwungen, identische Namen für Elemente Strukturen zu vergeben, so müssen wir auch eine Möglichkeit besitzen, die es uns gestattet, uns eindeutig auf ein bestimmtes Strukturelement zu beziehen. Wir erreichen das durch die Verwendung sogenannter gekennzeichneter Namen. Ein gekennzeichneter Name sieht wie folgt aus:

$$\underline{h}.\underline{u_1}.\underline{u_2}.\cdots.\underline{u_n}.\underline{e}$$

Hierbei bedeuten:

$\underline{h}$	Name der Hauptstruktur
$\underline{u_1}$ bis $\underline{u_n}$	Namen von Unterstrukturen
$\underline{e}$	Name des Strukturelementes

Beziehen wir uns auf obiges Beispiel, so können wir die aus der Auflösung der Option BY NAME resultierende Ergibtanweisung auch wie folgt schreiben:

```
J.L = M.L ;
```

Bei einer mehrfach gegliederten Struktur genügt es, in die gekennzeichneten Namen nur soviele Teilnamen aufzunehmen, wie zur eindeutigen Charakterisierung des Elementes erforderlich sind. Verwendet man alle, so spricht man von einem voll gekennzeichneten Namen.

Bei Verwendung eines gekennzeichneten Namens dürfen die Indizes jedem beliebigen Teilnamen angehängt werden, aber auch mehreren Teilnamen. So kann z.B. der gekennzeichnete Name

```
A.B(2).G(3)
```

auch durch

```
A.E.G(2,3)
```

ausgetauscht werden. Noch einfacher wäre der Name

```
A.G(2,3)
```

12.13. Abbildungen

In PL/1 ist es möglich, Typ und Form von Daten mittels einer Abbildung zu beschreiben. Eine solche Beschreibung tritt dann an die Stelle der sonst üblichen Beschreibung durch einzelne Attribute, wie z.B. CHAR(5). Um ein erstes Beispiel aufzuzeigen, die Abbildung '999V99' kann dazu benutzt werden, um ein Feld zu beschreiben, das aus drei Dezimalziffern besteht, einem gedachten Dezimalpunkt und zwei weiteren Dezimalziffern. Dieses Feld belegt also genau fünf Stellen, da der gedachte Dezimalpunkt im Feld nicht explizit erscheint. Eine solche Abbildung ist im DECLARE-Statement an Stelle der Attribute FIXED, CHARACTER, BIT usw. zu benutzen. Alle durch eine Abbildung beschriebenen Daten zeichnen sich dadurch aus, daß jedes in ihnen enthaltene Zeichen gerade eine Stelle (ein Byte) belegt.

Eine Abbildung kann entweder eine Zeichenkette repräsentieren oder ein numerisches Feld. Arithmetische Daten können nur auf Dezimalbasis durch eine Abbildung erfaßt werden, freilich allerdings sowohl in der Festpunkt- als auch in der Gleitpunktdarstellung. Komplexe Daten müssen durch eine einzige Abbildung repräsentiert werden,

die dann sowohl für den Realteil als auch für den Imaginärteil gilt. Außerdem ist noch das Attribut COMPLEX anzuführen. Für Dualdaten können Abbildungen nicht spezifiziert werden.

Jede Abbildung muß den Variablen in der DECLARE-Vereinbarung mit dem PICTURE-Attribut (Abkürzung: PIC) beigestellt werden. Eine solche Erklärung sieht also wie folgt aus:

```
DCL  v PICTURE'abbildung' ;
```

Jede Abbildung setzt sich aus einzelnen Zeichen, den sogenannten Abbildungszeichen, zusammen.

Wir wollen uns zunächst mit den Abbildungen beschäftigen, die für Zeichenkettendaten gelten. Für diese dürfen die Abbildungszeichen benutzt werden, die in der Abb. 12.18. dargestellt sind.

Abbildungszeichen	Bedeutung
A	Die entsprechende Stelle der Zeichenkette darf irgendeinen Buchstaben oder das Leerzeichen enthalten.
X	Die entsprechende Stelle der Zeichenkette darf irgendein beliebiges Zeichen enthalten.
9	Die entsprechende Stelle der Zeichenkette darf irgendeine Dezimalziffer oder das Leerzeichen enthalten.

Abb. 12.18. Abbildungszeichen für Zeichenkettendaten

Innerhalb der Abbildung muß mindestens eines der Abbildungszeichen X oder A vorkommen.

Wenn einer Variablen, die durch eine Zeichenkettenabbildung beschrieben ist, ein Wert zugewiesen wird, so wird jedes Zeichen dieses Wertes geprüft. Wenn dabei festgestellt wird, daß das Datenzeichen nicht mit dem entsprechenden Abbildungszeichen übereinstimmt, so wird die CONVERSION-Bedingung gesetzt. An einem simplen Beispiel sei das in der Abb. 12.19. demonstriert.

```
DECLARE  V1 CHARACTER(4),
         V2 PICTURE'A99X' ;
W1 = 'L45%' ;
W2 = W1 ;
:
W2 = 'J5R4' ;
```

Abb. 12.19. Beispiel für eine durch das PICTURE-Attribut beschriebene Zeichenkettenvariable

Bei der letzten Ergibtanweisung würde die CONVERSION-Bedingung auftreten, da das dritte Zeichen, d.h. R, der Zeichenkettenkonstanten auf der rechten Seite der Ergibtanweisung nicht mit dem entsprechenden Abbildungszeichen, d.h. 9, übereinstimmt.

Wir wollen uns nun mit den Abbildungszeichen befassen, die zur Beschreibung numerischer Datenfelder benutzt werden können (Abb. 12.20.).

Abbildungszeichen	Bedeutung bzw. echtes Zeichen
9	Beliebige Dezimalziffer
V	Kein echtes Zeichen; V steht für einen gedachten Dezimalpunkt. Beispiel: Wollen wir, daß eine Variable, die den Wert 234,56 enthält, zu 23456, d.h. ohne Dezimalpunkt ausgeben, so müssen wir die Abbildungszeichen 999V99 spezifizieren. Das V steht also für kein Zeichen im Feld, es gibt bloß an, wo der Dezimalpunkt angenommen werden soll.
S	Vorzeichenposition im Feld; es erscheint ein Pluszeichen, wenn der Wert ≧ 0 ist, ein Minuszeichen, wenn der Wert < 0 ist.
+	Vorzeichenposition im Feld; es erscheint ein Pluszeichen, wenn der Wert ≧ 0 ist, anderenfalls ein Leerzeichen.
-	Vorzeichenposition im Feld; es erscheint ein Minuszeichen, wenn der Wert < 0 ist, anderenfalls ein Leerzeichen.
E	Es erscheint der Buchstabe E, der den Beginn des Exponententeilfeldes anzeigt (bei Gleitpunktdarstellungen)
Z	Durch dieses Zeichen wird eine Ziffernstelle angezeigt, die von einem Leerzeichen belegt wird, wenn ansonsten auf sie eine führende Null fallen würde. Somit wird 00123 zu bb123 aufbereitet, wenn die Abbildung durch die Zeichenfolge ZZZZZ bewirkt wird[2]).
*	Dieses Zeichen bewirkt das gleiche wie Z, nur daß führende Nullen durch das Sternzeichen ersetzt werden.
Y	Dieses Zeichen bewirkt das gleiche wie Z, nur daß alle Nullen durch Leerzeichen ersetzt werden. Somit wird bei Vorliegen der Abbildung Y9/Y9/Y9 aus der Kette 08/05/67 die Zeichenfolge b8/b5/67 hergestellt[2]).
$	Ein Währungszeichen erscheint auf der angegebenen Zeichenposition. Dieses Zeichen ist nur gültig, wenn es auf der ersten oder letzten Stelle eines Teilfeldes angegeben wird.
,	Ein Komma wird in die gekennzeichnete Position eingefügt. Ist vor dem Komma ein Z oder ein Sternzeichen aufgeführt und ruft dieses eine Nullenunterdrückung hervor, so erscheint auf der durch das Komma spezifierten Stelle ebenfalls ein Leerzeichen oder ein Sternzeichen, wenn links vom Komma keine gültigen Ziffern auftreten.
.	Die gleiche Bedeutung wie beim Komma liegt diesem Zeichen zugrunde, nur daß ein Punkt auftritt.
/	Die gleiche Bedeutung wie beim Komma liegt diesem Zeichen zugrunde, nur daß ein Schrägstrich auftritt.
B	Ein Leerzeichen erscheint in der angegebenen Stelle.
T	Beliebige Dezimalziffer, die eine Vorzeichenüberlochung aufweist (12-er Lochung bei positivem Wert, 11-er Lochung bei negativem Wert)
I	Beliebige Dezimalziffer, die eine Vorzeichenüberlochung aufweist, wenn der Wert ≧ 0 ist
R	Beliebige Dezimalziffer, die eine Vorzeichenüberlochung aufweist, wenn der Wert < 0 ist.
CR	Es erscheinen die beiden Zeichen CR, wenn der Wert < 0 ist.
DB	Wie CR, nur daß die beiden Zeichen DB erscheinen.
F	Kein echtes Zeichen. Dieses Abbildungszeichen gibt vielmehr an, daß ein Skalenfaktor in der Abbildung folgt; dieser ist in Klammern einzuschließen und kann positiv oder negativ sein. Ein Skalenfaktor bewirkt, daß der im Feld stehende Wert mit der Potenz multipliziert wird, deren Basis 10 und deren Exponent der Skalenfaktor ist, bevor er in Operationen eingeht.

Anmerkung: b bedeutet hier das Leerzeichen.

Abb. 12.20. Abbildungszeichen für numerische Datenfelder

Mit Ausnahme von Variablen in Gleitpunktdarstellung darf innerhalb einer Abbildung nur ein einziges Abbildungszeichen zur Festlegung von Vorzeichen benutzt werden. Dieser Einschränkung unterliegen also die Abbildungszeichen S, +, -, T, I, R, CR und DB. Bei Variablen in Gleitpunktdarstellung darf jedes Teilfeld (Dezimalbruch und Exponent) sein eigenes Abbildungszeichen zur Festlegung der Vorzeichenstelle innerhalb des Teilfeldes aufweisen. Beim Entwurf von Abbildungen ist zusätzlich zu beachten, daß die Abbildungszeichen für Vorzeichen nur ganz bestimmten Stellen der Felder bzw. Teilfelder der Variablen entsprechen dürfen. Es gelten folgende Regeln:

a) S, + und - dürfen nur links oder rechts von allen Abbildungszeichen erscheinen, die Ziffernstellen repräsentieren.

b) T, I, R, CR und DB dürfen nur rechts von allen Abbildungszeichen erscheinen, die Ziffernstellen repräsentieren.

Die Wiederholung eines Abbildungszeichens kann dadurch erreicht werden, daß vor das zu wiederholende Abbildungszeichen ein Wiederholungsfaktor geschrieben wird. Dieser ist in Klammern einzuschließen.

Bekanntlich sind die Abbildungen von Anführungszeichen zu umgeben und mittels des PICTURE-Attributes den Variablen zuzuordnen. Auf dieser Basis wollen wir auch die nunmehr folgenden Beispiele betrachten (Abb. 12.21.).

```
DECLARE  H DECIMAL FIXED(5,3),
         K PICUTRE'abbildung' ;
         K = H ;
```

Wert von H	abbildung	Wert von K
87.587	999V99	Ø8758
87.587	(3)9V9T	Ø875H
87.587	-999F(-1)	b875
87.587	9999CR	ØØ87bb
87.587	-999ES99	b876E-Ø1
87.587	+V999KS99	+876+Ø2

Anmerkung: b bedeutet hier das Leerzeichen

Abb. 12.21. Beispiele von Abbildungen für numerische Datenfelder

Für Gleitpunktfelder haben wir noch einen Hinweis zu geben: Das Abbildungszeichen F darf für sie nicht spezifiziert werden.

Die Abbildungszeichen

Z * $, . / B

sind sicher in erster Linie für die Aufbereitung von zu druckenden Daten gedacht. Sehen wir uns dazu das Beispiel

```
DECLARE  DF FIXED(5,2) INITIAL(-4,6),
         DG PIC'***V.99CR' ;
DG = DF ;
```

an. Bevor der Wert von DF der Variablen DG zugewiesen wird, wird er gemäß der für DG festgelegten Abbildung aufbereitet, wodurch

**4.6ØCR

erhalten wird. Die umgekehrte Zuweisung, praktiziert durch die Ergibtanweisung

```
DF = DG ;
```

ist selbstverständlich ebenfalls durchführbar. Doch verfolgen wir einmal bei diesem letzten Beispiel die Einzelschritte, um die einzelnen Phasen der Umwandlung kennenzulernen. Zunächst wird der Wert von DG in eine Dezimalzahl in Festpunktdarstellung mit der für DG spezifizierten Genauigkeit gebracht. Um das zu erreichen, wird ein Unterprogramm aufgerufen, das die beiden führenden Sternzeichen entfernt und die Zeichenfolge CR durch das Minuszeichen ersetzt. Nach diesem ersten Schritt kann dann der so erhaltene Wert gepackt und schließlich DF zugewiesen werden. Aus dieser Betrachtung heraus haben wir praktisch nebenbei auch die Erkenntnis gewonnen, daß Variable, deren Eigenschaften durch das PICTURE-Attribut mit einer Abbildung beschrieben sind, ebenfalls in arithmetische Ausdrücke aufgenommen werden dürfen. Unnötige Umwandlungen numerischer durch Abbildungen beschriebener Felder sollten jedoch weitgehend vermieden werden, da es sich bei ihnen um relativ langsame Computeroperationen handelt, besonders wenn Abbildungen vorliegen, die andere Abbildungszeichen als das Zeichen 9 aufweisen.

Treten die Abbildungszeichen nur ein einziges Mal in einer Abbildung auf, so bezeichnet man sie als statische Abbildungszeichen und ihre Bedeutung kann der Abb. 12.20. entnommen werden. Gemäß ihrer Funktion (Unterdrückung führender Nullen) werden die beiden Abbildungszeichen Z und * nur in den führenden Stellen eines Feldes oder eines Teilfeldes verwendet, und zwar gewöhnlich zu mehreren nebeneinander. Vier andere Zeichen,nämlich

```
$ S + -
```

können ebenfalls zu mehreren nebeneinander auf den führenden Stellen einer Abbildung angegeben werden. Erscheinen sie auf diese Weise als Zeichenfolge, so werden sie als Driftzeichen bezeichnet. Driftzeichen haben folgende Bedeutung: Unmittelbar vor die erste von Null verschiedene Ziffer eines Feldes werden das Währungszeichen oder das Pluszeichen oder das Minuszeichen oder ein Leerzeichen in das Feld gestellt. Tritt keine von Null verschiedene Ziffer in den durch diese Zeichen beschriebenen Positionen auf, so erscheint das entsprechende Driftzeichen auf der äußeren rechten Stelle, auf die die Folge der Driftzeichen verweist. Durch diese Verwendung der Symbole $, S, + - wird also bewirkt, daß die führenden Ziffern eines Feldes durch Leerzeichen ersetzt werden, falls sie gleich Null sind. Die Aufbereitung erfolgt also gerade so, als ob das Abbildungszeichen Z niedergeschrieben worden wäre, nur daß das Währungszeichen oder evtl. ein Vorzeichen auf der Stelle erscheint, auf der sonst das letzte Leerzeichen auftreten würde.

Wir wollen der Übersicht wegen für Driftzeichen in Abbildungen einige typische Beispiele betrachten (Abb. 12.22.).

Einige abschließende Bemerkungen sind noch angebracht. Wird ein Driftzeichen rechts vom V verwendet, so muß es alle Ziffernstellen rechts vom V kennzeichnen. Ist der Wert gleich Null, so besetzen Sternzeichen bzw. Leerzeichen alle Stellen des aufbereiteten Feldes. Im anderen Falle werden alle rechts vom V stehenden Ziffern als gültig betrachtet, das V spielt also bei der Aufbereitung gewissermaßen die Rolle einer gültigen Ziffer. Es ist weiterhin zu beachten, daß das alleinige Niederschreiben des Dezimalpunktes nicht genügt, um die Stelle zu kennzeichnen, die als Dezimalstelle interpretiert werden soll. Somit wird bei der Aufbereitung von Ø.Ø6 ein leeres Feld entstehen, wenn die Abbildung kein V enthält. Der Kompilierer nimmt hierbei an, daß der Dezimalpunkt rechts vom letzten Abbildungszeichen auftritt; damit wird aber der Dezimalteil des Wertes im aufbereitenden Feld unterdrückt.

Wert	Abbildung	Ergebnis
6328.24	$$$,$$$.	b$6,328.
6328.24	$$$. (3)$,	b$6.328,
6328.24	(5)ZV.99F(-2)	bbb63.28,
-321.Ø6	--,---V.9	bb-321.Ø
-321.Ø6	ZZZZZ.99	bbbb3.21
-321.Ø6	-V.999KS99	-.321+Ø3
Ø.Ø6	+ZZZZV.99	+bbbb.Ø6
Ø.Ø6	+++++V.99	bbbb+.Ø6
Ø.Ø6	ZZZ99V.99	bbbØØ.Ø6
Ø.ØØ	$***V.** -	$******b
7Ø471	B9/Y9/99	b7/b4/71

Anmerkung: b bedeutet hier das Leerzeichen

Abb. 12.22. Beispiele für Driftzeichen in Abbildungen

12.14. Die satzweise Eingabe/Ausgabe

Bisher haben wir die drei Arten der reihenweisen Eingabe/Ausgabe, nämlich die listengesteuerte, die datengesteuerte und die formatgesteuerte kennnengelernt. Allen drei Arten ist letzten Endes gemeinsam, daß die Datenanordnung auf den Eingabe/Ausgabemedien als eine Reihe von Zeichen angesehen wird, die in irgendeiner Art und Weise voneinander getrennt sind. Bei der Eingabe wird eine festumrissene Teilfolge dieser Zeichen in die interne Form umgewandelt, die für das betreffende Datenelement erklärt ist. Bei der Ausgabe läuft der umgekehrte Vorgang ab.

Neben dieser reihenweisen Eingabe/Ausgabe ist in PL/1 außerdem die sogenannte satzweise Eingabe/Ausgabe unterstützt. Der satzweisen Eingabe/Ausgabe legt man folgendes Gedankengut zugrunde: Man denkt sich die Datenanordnung auf den externen Medien so aufgebaut, daß die Informationen zu einzelnen voneinander abgesonderten Sätze zusammengefaßt sind, deren Darstellung ein naturgetreues Abbild der internen Darstellung ist. Damit wird die Eingabefunktion zurückgeführt auf die Kopierung eines Satzes vom externen Medium auf den Hauptspeicher; unter der Ausgabe wäre dann sinngemäß die Kopierung eines Satzes im Hauptspeicher auf das externe Medium zu verstehen. Da ein Satz so betrachtet wird, als ob er die Daten in der Form enthält, die der Strukturerklärung zugrundeliegt, erfolgt bei der Eingabe bzw. der Ausgabe keine Datenumwandlung. Das bedeutet aber, daß sich der Programmierer um die Codierung von Datenumwandlungen kümmern muß, falls doch welche bei der Eingabe oder der Ausgabe vorgesehen werden müssen. Betrachtet man die Zeit, die bei der reihenweisen Eingabe/Ausgabe für die Datenumwandlungen erforderlich ist, so kann man ohne weiteres behaupten, daß die satzweise Eingabe/Ausgabe gewöhnlich schneller abläuft.

Für die Dateien, die in die satzweise EA eingehen sollen, müssen selbstverständlich beschreibende Elemente vorliegen. Bei ihnen muß man zwischen denen, die abhängig von der Maschinen- bzw. Programmausrüstung (Hardware bzw. Software) sind, und solchen unterscheiden, die anwendungsmäßig festgelegt werden müssen. Die Quelle für die ersteren Aussagen sind die relevanten Steuerstatements der Jobsteuersprachen, also

—— im DOS die Statements TLBL, DLBL und EXTENT und
—— im OS das DD-Statement.

Anwendungsabhängige Angaben erfolgen einmal in der DECLARE-Vereinbarung und zum anderen in der OPEN-Anweisung.

Sind nicht alle notwendigen Eigenschaften einer Datei in irgendeiner Weise beschrieben worden, so tritt in gewissen Fällen eine Standardeigenschaft in Kraft.

Den nachfolgenden Erörterungen legen wir die folgende DECLARE-Vereinbarung zugrunde:

```
DECLARE  name attribute ;
```

Eine Datei mit dem Namen name kann dann durch die nachstehenden Attribute charakterisiert werden:

a) FILE
Dieses Attribut kann aber muß nicht erklärt werden; bei Benutzung anderer Dateiattribute ist name implizit als Dateiname festgelegt.

b) STREAM | RECORD
Durch diese Attribute wird die Art der EA festgelegt; STREAM bedeutet die reihenweise EA, RECORD die satzweise. Fehlt eines dieser beiden Attribute, so wird standardmäßig STREAM angenommen. Beide Attribute schließen automatisch auch FILE ein.

c) PRINT
Durch dieses Attribut wird eine Druckdatei festgelegt. Dieses Attribut schließt die Attribute OUTPUT und FILE ein, so daß diese nicht zusätzlich erklärt zu werden brauchen, aber natürlich erklärt werden können.

d) INPUT | OUTPUT | UPDATE
Diese Attributgruppe legt fest, wie eine Datei verwendet werden soll. INPUT spezifiziert eine Eingabedatei, OUTPUT eine Ausgabedatei und UPDATE eine Datei, von der sowohl Daten eingelesen als auch in die Daten ausgegeben werden sollen [1]. Standardattribut ist INPUT; es wird immer dann zugeordnet, wenn ein Attribut aus dieser Gruppe fehlt.

Die Angabe von UPDATE schließt automatisch auch die Angabe von RECORD ein, d.h. die Aufdatierung von Dateien ist nur mittels der satzweisen EA möglich.

e) SEQUENTIAL | DIRECT
Durch diese Attribute wird die Zugriffsmethode festgelegt. Beide Attribute schließen RECORD ein, DIRECT zusätzlich auch KEYED. Der Einsatz der direkten Zugriffsmethode verlangt kategorisch die Erfüllung bestimmter gerätetechnischer Gegebenheiten: Speichergeräte mit direkter Zugriffsmöglichkeit, wie Magnetplattenspeichergeräte usw.

Standardattribut dieser Gruppe ist SEQUENTIAL.

f) BUFFERED | UNBUFFERED
Durch diese Attribute wird die Verwendung von Puffern bei der EA geregelt. Beide Attribute schließen das Attribut RECORD ein. Die Pufferung der EA wird automatisch bei der sequentiellen Zugriffsmethode unterstellt, wenn diese Attributgruppe in der Dateierklärung nicht vertreten ist. Liegt die direkte Zugriffsmethode vor und fehlt das Pufferungsattribut, so wird die ungepufferte Arbeitsweise, d.h. das Attribut UNBUFFERED

1) UPDATE erfordert deshalb gerätetechnische Voraussetzungen: nur Speichergeräte mit direkter Zugriffsmöglichkeit (Magnetplattenspeichergeräte etc.) können für eine solche Datei bereitgestellt werden.

automatisch zugeordnet. Die ungepufferte Arbeitsweise werden wir in Abschnitt 13.2. behandeln.

g) BACKWARDS
Dieses additive Dateiattribut schließt gleichzeitig die Attribute RECORD, INPUT und SEQUENTIAL ein und bedeutet, daß die Sätze rückwärts von der Datei gelesen werden, d.h. der physische letzte Satz wird als erster und der physisch erste Satz als letzter gelesen. Ein solches Vorgehen erfordert ebenfalls die Erfüllung bestimmter gerätetechnischer Voraussetzungen: Magnetbandgeräte sind erforderlich.

h) KEYED
Dieses Attribut zeigt an, daß die Datensätze durch Schlüssel ausgezeichnet sind. Es schließt gleichzeitig das Attribut RECORD ein.

i) EXCLUSIVE
Dieses Attribut besagt, daß die Sätze einer aufzudatierenden Datei, zu denen direkt zugegriffen wird, gegenüber den Zugriffen eines anderen Programmes solange gesperrt sind, bis die Operation abgeschlossen ist. Dieses Attribut schließt die Attribute RECORD, UPDATE, DIRECT und KEYED ein. Die Datei, auf die es bezogen ist, muß auf einem Speichergerät mit direkter Zugriffsmöglichkeit liegen, d.h. auf Magnetplattenspeichern usw.

Dieses Attribut kann nur in PL/1-Programmen benutzt werden, die unter dem Betriebssystem OS laufen.

j) INTERNAL | EXTERNAL
Diese beiden Attribute legen den Gültigkeitsbereich des Dateinamens fest. Fehlt eines dieser Attribute, so gilt für den Dateinamen standardmäßig das Attribut EXTERNAL.

k) ENVIRONMENT(angaben)
Durch dieses Attribut werden hard- und softwarespezifische Angaben erfaßt. Im Betriebssystem OS ist dieses Attribut nur dann erforderlich, wenn einige unumgänglich notwendige Angaben vorliegen müssen. Im Betriebssystem DOS ist hingegen die Aufführung dieses Attributes unvermeidbar, und zwar wegen der Angabe MEDIUM.

Die einzelnen Angaben sind nebeneinander zu stellen und durch Leerzeichen voneinander zu trennen. Die Aufzählung aller Angaben ist schließlich in Klammern einzuschließen.

Folgende Angaben können getätigt werden:

—— F | FB | V | VB | U | FS | FBS | VS | VBS

Durch diese Angabe kann das Satzformat festgelegt werden. Für das Betriebssystem DOS sind nur die ersten fünf Möglichkeiten gegeben

—— RECSIZE(satzlänge)
—— BLKSIZE(blocklänge)
—— BUFFERS(n)

Durch diese Angabe wird die Anzahl n der EA-Gebiete (Puffer) festgelegt. Im Betriebssystem DOS kann n nur die Werte 1 oder 2 annehmen, im Betriebssystem OS muß n kleiner als 256 sein. Fehlt diese Angabe, so wird für n standardmäßig 2 angenommen.

—— CONSECUTIVE | INDEXED | VSAM |
REGIONAL(1) | REGIONAL(2) | REGIONAL(3)

Durch diese Gruppe von Angaben wird die Organisationsform der Datenmengen festgelegt. CONSECUTIVE bedeutet die konsekutive (starr fortlaufende) Organisations-

form; diese Angabe wird bei Fehlen einer Angabe dieser Gruppe als Standardangabe unterstellt. INDEXED bedeutet die indexsequentielle (logisch fortlaufende), VSAM die Organisationsform VSAM und die verschiedenen REGIONAL-Angaben die regionale Organisationsform. Die Angabe REGIONAL(2) ist im Betriebssystem DOS nicht zulässig.

—— LEAVE | REREAD | UNLOAD

Durch eine dieser Angaben wird die Disposition von Magnetbändern festgelegt. Näheres hierzu ist im Abschnitt 11.10. besprochen worden.

—— CTLASA | CTL36Ø

Diese Angaben können nur für Ausgabedateien vorgenommen werden. Sie legen fest, daß das erste Zeichen eines Satzes ein Gerätesteuerzeichen ist (Fachauswahl bei Lochkartenstanzern, Vorschub bzw. Zeilentransport bei Zeilendruckern).

—— TOTAL

Diese Angabe ist nur im Betriebssystem OS erlaubt. Durch sie wird festgelegt, daß der Kompilierer den effektivsten Maschinencode produzieren kann. Das aber setzt voraus, daß alle Dateiattribute und -angaben im DECLARE-Statement erscheinen müssen. Es dürfen weder Angaben im DD-Steuerstatement der Jobsteuersprache erfolgen, noch dürfen Dateiattribute erst beim Eröffnungsprozeß (OPEN-Anweisung) zugeordnet werden. Diese Angabe beschleunigt viele Programme beträchtlich.

—— COBOL

Diese Angabe besagt, daß die mit der betreffenden Datei benutzten Strukturen das Aussehen von Strukturen in COBOL haben. Dadurch wird der Datenaustausch mit Programmen, die in COBOL geschrieben sind, erheblich erleichtert.

—— INDEXAREA [(gebietsumfang)]

Durch diese Angabe wird festgelegt, daß das Hauptinhaltsverzeichnis einer indexsequentiell organisierten Datenmenge so weit wie möglich im Hauptspeicher gehalten wird. Im Betriebssystem DOS muß der Umfang des hierfür bereitzustellenden Gebietes in Bytes angegeben werden.

—— NOWRITE

Durch diese Angabe wird festgelegt, daß keine Sätze zu einer Datenmenge hinzugefügt werden, die die Attribute DIRECT und UPDATE aufweist. Diese Angabe bewirkt die Erhöhung des Durchsatzes innerhalb einer bestimmten Zeit.

—— ADDBUFF [(n)]

Bei Dateien, für die der direkte Zugriff vorgesehen ist, wird durch diese Angabe der Umfang n in Bytes eines Gebietes festgelegt, das zur Neuanordnung der Daten bei der EA benutzt wird. Dieses Gebiet muß beim Betriebssystem DOS mindestens so groß sein, wie die im Z4 vermehrte Summe aus Blocklänge und Schlüssellänge. Im Betriebssystem OS kann n nicht spezifiziert werden; es wird stets für das Gebiet eine Größe unterstellt, die gleich dem Spurumfang des benutzten externen Speichergerätes ist. Diese Angabe erhöht den Durchsatz auf Kosten zusätzlichen Hauptspeicherplatzes.

—— GENKEY

Durch diese Angabe wird festgelegt, daß Datenmengen, deren Organisationsform durch VSAM oder INDEXED beschrieben ist, generische Schlüssel aufweisen.

—— TRKOFL

Diese Angabe kann nur im Betriebssystem OS erfolgen. Sie besagt, daß die Spurenüberlaufeinrichtung bei Magnetplattenspeichern benutzt wird. Diese Angabe kann nicht auf die Organisationsformen INDEXED und REGIONAL(3) bezogen werden.

—— NCP(n)

Diese Angabe entspricht dem entsprechenden Parameter des DD-Statements; sie kann nur für das Betriebssystem OS getätigt werden.

—— KEYLENGTH(n)

Durch diese Angabe wird die Schlüssellänge n in Bytes festgelegt, die bei Datenmengen der Organisationsform INDEXED und REGIONAL(3) benutzt werden.

—— KEYLOC(n)

Durch diese Angabe wird die Stelle eines Satzes festgehalten, auf der der in den Satz eingebettete Schlüssel beginnt. Diese Angabe ist bei der Erstellung einer indexsequentiell organisierten Datenmenge erforderlich. Im Betriebssystem DOS muß diese Angabe für geblockte Sätze getätigt werden, im Betriebssystem OS hingegen genügt an ihrer Stelle auch die Aufführung des RCP-Parameters im DD-Steuerstatement der Jobsteuersprache.

—— SCALARVARYING

Diese Angabe ist im Zusammenhang mit Zeichenketten variabler Länge zu benutzen. Über diese haben wir noch nicht gesprochen.

Die nachfolgenden Angaben gelten prinzipiell nur für das Betriebssystem DOS.

—— MEDIUM(SYSxxx [,gerätetyp])

Diese Angabe ist stets zwingend vorgeschrieben. Der Gerätetyp ist bei der satzweisen EA stets aufzuführen. Einzelheiten über diese Angabe können dem Abschnitt 11.10. entnommen werden.

—— ASSOCIATE(dateiname)

Durch diese Angabe wird für eine Datenmenge, die einem Lochkartengerät des Typs IBM 3525 zugeordnet ist, eine assoziierte Datei festgelegt, wodurch es bei der satzweisen EA ermöglicht wird, daß zwei Operationen, wie Lesen und Stanzen, durchgeführt werden können. Dabei müssen die Operationen in der Reihenfolge Lesen, Stanzen, Schreiben, Lesen angesprochen werden.

—— RCE

Durch diese Angabe wird für eine Datenmenge, die einem Lochkartengerät der Typen IBM 3505 oder IBM 3525 zugeordnet ist, festgelegt, daß selektives Lesen vorgenommen werden soll.

—— OMR

Durch diese Angabe wird für eine Datenmenge, die einem Lochkartengerät der Typen IBM 3505 oder IBM 3525 zugeordnet ist, festgelegt, daß optisches Markierungslesen erfolgen soll.

—— COLBIN

Durch diese Angabe wird für eine Datenmenge, die einem Lochkartengerät der Typen IBM 3505 oder IBM 3525 zugeordnet ist, festgelegt, daß Dualkartenverarbeitung erfolgen soll.

—— STACKER(n)

Durch diese Angabe wird für eine Datenmenge, die einem Lochkartengerät der Typen IBM 3505 oder IBM 3525 zugeordnet ist, festgelegt, in welches Ablagefach die Lochkarten abgelegt werden sollen. Für n kann 1 oder 2 aufgeführt werden, standardmäßig wird 1 angenommen.

—— NOLABEL

Durch diese Angabe wird festgelegt, daß die Datenmenge auf einem Magnetband ohne Kennsätze (Etiketten) residiert.

—— NOTAPEMK

Durch diese Angabe wird festgelegt, daß die Datenmenge auf einem Magnetband ohne Kennsätze (Etiketten) residiert und daß keine führende Bandmarke besitzt.

—— VERIFY

Durch diese Angabe wird festgelegt, daß alle auf eine SEDZ [1] geschriebenen Sätze für Prüfzwecke noch einmal gelesen werden.

—— EXTENTNUMBER(n)

Durch diese Angabe wird die Anzahl der Bereiche festgelegt, die eine Datenmenge auf SEDZ belegt. Bei der regionalen Organisationsform wird bei Fehlen dieser Angabe als Standard 1 angenommen, bei der indexsequentiellen als Standard 2.

—— INDEXMULTIPLE

Durch diese Angabe wird festgelegt, daß ein Hauptinhaltsverzeichnis für diese Datenmenge benutzt wird (indexsequentielle Organisationsform).

—— HIGHINDEX(typ)

Durch diese Angabe kann der Gerätetyp für das Hauptinhaltsverzeichnis bei einer indexsequentiell organisierten Datenmenge festgelegt werden (2311, 2314 usw.), wenn das hierfür vorgesehene Gerät typmäßig von dem für die Datenmenge in der MEDIUM-Angabe vorgesehenen Gerät abweicht.

—— OFLTRACKS(n)

Durch diese Angabe wird bei der Erstellung von indexsequentiell organisierten Datenmengen festgelegt, wieviele Spuren pro Zylinder für Überlaufsätze bereitgestellt werden sollen.

Eine Datei muß dann und nur dann erklärt werden, wenn ein Attribut ihr unbedingt zugeordnet werden muß. Da im Betriebssystem DOS die Angabe MEDIUM im ENVIRONMENT-Attribut vorhanden sein muß, müssen in Programmen, die unter dem Betriebssystem DOS laufen sollen, die Dateien auch in DECLARE-Vereinbarungen erklärt werden.

1) SEDZ ist die Abkürzung von Speichereinheit mit direkter Zugriffsmöglichkeit.

Bevor in eine Datei geschrieben bzw. von einer Datei gelesen werden kann, muß die Datei eröffnet werden. Das Eröffnen einer Datei veranlaßt das Betriebssystem, einige Steuerbits zu setzen und die Kennsätze (bei Magnetbändern bzw. Magnetplatten als Datenträger) zu verarbeiten. Eine Datei kann auf zwei Arten eröffnet werden:

a) durch Verwendung der OPEN-Anweisung (explizites Eröffnen),
b) implizit.

Man sagt, daß eine Datei implizit eröffnet wird, wenn ein GET/PUT- oder ein anderes Eingabe/Ausgabestatement, das sich auf die betreffende Datei bezieht, ausgeführt wird, ohne daß sie zuvor dem Eröffnungsprozeß unterworfen wurde. Das implizite Eröffnen einer Datei erlaubt dem Programmierer natürlich nicht, irgendwelche Attribute noch zur Eröffnungszeit einer Datei zuzuordnen.

Die OPEN-Anweisung besitzt bekanntlich folgende allgemeine Form:

```
OPEN FILE(dateiname) [optionen]
     [,FILE(dateiname) [optionen]] ... ;
```

Aus der allgemeinen Form dieses Statements kann man erkennen, daß mehr als eine Datei zur gleichen Zeit eröffnet werden kann. Als Optionen können mit Ausnahme von INTERNAL, EXTERNAL und ENVIRONMENT alle anderen Attribute angegeben werden. Auf diese Weise ist es möglich, bestimmte Eigenschaften von Dateien erst zur Eröffnungszeit festzulegen. Werden Attribute einem Dateinamen auf verschiedene Arten (DECLARE-Vereinbarung, OPEN-Anweisung, standardmäßig) zugeordnet, so dürfen sich diese selbstverständlich unter keinen Umständen widersprechen.

Nachdem einmal eine Datei eröffnet worden ist, können wir ihre Sätze lesen, indem wir das Statement

```
READ FILE(dateiname) INTO(variable) ;
```

benutzen. Hierbei ist unter variable der Name einer nichtindizierten Hauptstruktur zu verstehen, d.h. einer Struktur, der die Stufennummer 1 zugeordnet ist. Man beachte, daß für die reihenweise Eingabe das Schlüsselwort GET und für die satzweise Eingabe das Schlüsselwort READ zu verwenden ist. Das Statement

```
READ FILE(dateiname) [IGNORE(ausdruck)] ;
```

bewirkt zunächst die Berechnung von ausdruck und Umwandlung desselben in eine ganze Zahl. Ist diese positiv, so werden genau so viele Sätze der Datei übergangen, wie diese ganze Zahl angibt, d.h. sie werden nicht dem Programm zur Verfügung gestellt. Ist die gefundene ganze Zahl negativ, so wird dieses Statement nicht beachtet. Schreibt man einfach

```
READ FILE(dateiname) ;
```

so daß es später in der Lage ist, die betreffende Datei dem geeigneten Eingabe/Ausgabe-

Die satzweise Ausgabe wird durch das Statement

```
WRITE FILE(dateiname) FROM(variable) ;
```

veranlaßt. Auch hier ist unter variable der Name einer nichtindizierten Hauptstruktur zu verstehen.

Eine Datei wird automatisch, d.h. implizit, eröffnet, wenn sie nicht eröffnet ist, und das erste READ- bzw. WRITE-Statement für diese Datei ausgeführt wird. READ schließt gleichzeitig das Attribut INPUT und WRITE das Attribut OUTPUT ein, so daß diese Attribute in einem solchen Fall nicht besonders erklärt werden müssen.

Nach Verarbeitung des letzten Satzes muß eine Datei auch abgeschlossen werden, d.h. es müssen Routinen des Betriebssystems ablaufen, die die Datei als nicht mehr verfügbar kennzeichnen und Kennsätze schreiben bzw. prüfen (bei Magnetbändern und Platt enspeichern). Für diese Zwecke ist das Statement

```
CLOSE FILE(dateiname) [,FILE(dateiname)] ... ;
```

einzusetzen. Ein CLOSE-Statement für eine nicht eröffnete Datei ist gänzlich ohne Wirkung; ebenso natürlich ein OPEN-Statement für eine bereits eröffnete Datei. Nachdem eine Datei abgeschlossen worden ist, kann sie wieder neu eröffnet werden, falls die Logik des Programmes eine derartige Maßnahme erfordert. Mit dem Abschließen verliert eine Datei auch die Eigenschaften, die ihr durch Attribute zugeordnet wurden. Sie erhält durch das Neueröffnen nur diejenigen Attribute zurück, die im DECLARE-Statement für sie angegeben sind. Dadurch ist es z.B. möglich, daß man eine Datei in einem Programm zunächst schreibt und anschließend sofort wieder liest. Fehlt das CLOSE-Statement, so werden eröffnete Dateien implizit abgeschlossen.

Auch die CLOSE-Anweisung kann wie die OPEN-Anweisung gleichzeitig mehrere Dateien ansprechen. Sie kann darüberhinaus auch noch folgende Gestalt annehmen:

```
CLOSE FILE(dateiname) [ENVIRONMENT(angabe)]
      [,FILE(dateiname) [ENVIRONMENT(angabe)]] ... ;
```

Bei dem hier als Option verwendeten Attribut können als Angabe nur LEAVE, REREAD (Betriebssystem OS) oder UNLOAD (Betriebssystem DOS) niedergeschrieben werden.

Am Programmende werden alle noch eröffneten Dateien automatisch abgeschlossen. Eine CLOSE-Anweisung ist also nicht unbedingt erforderlich.

Eine Option kann bei der OPEN-Anweisung zusätzlich spezifiziert werden, die kein Attribut darstellt und infolgedessen auch nicht in der DECLARE-Vereinbarung erscheinen darf. Sie heißt:

```
TITLE(ausdruck)
```

Der Ausdruck wird in eine Zeichenkette umgewandelt, die folgende Bedeutung besitzt:

a) Betriebssystem OS
   Die ersten acht Zeichen dieser Zeichenkette ersetzen den Dateinamen, d.h. sie werden als ddname des DD-Steuerstatements der Jobsteuersprache benutzt.

b) Betriebssystem DOS
   Die ersten sieben Zeichen dieser Zeichenkette werden als Name in den TLBL- oder DLBL-Steuerstatements der Jobsteuersprache benutzt.

Fehlt die TITLE-Option, so wird selbstverständlich der in der DCL-Vereinbarung, OPEN-Anweisung bzw. READ-/WRITE-Anweisungen verwendete Dateiname beibehalten und als ddname (OS) bzw. TLBL- oder DLBL-Name (DOS) benutzt. Da die für eine Datei gegebene OPEN-Anweisung bzw. die gegebenen OPEN-Anweisungen unterschiedliche Titel haben können, ist es möglich, zwei oder mehrere Datenmengen zu verarbeiten, ohne mehr als eine Datei im PL/1-Programm ansprechen zu müssen. Allerdings muß die Datei abgeschlossen werden, bevor sie für die nächste zu verarbeitende Datenmenge erneut eröffnet wird.

12.15. Behandlung eines Programmbeispieles für die satzweise EA

Um die satzweise EA besser verstehen zu können, wollen wir uns ein Programmbeispiel ansehen (Abb. 12.23.). Das Programmbeispiel soll dazu dienen, eine konsekutiv organisierte Datenmenge zu erstellen, deren Sätze Informationen über die USA-Kunden einer Handelsfirma enthalten. Aus zwei aufeinanderfolgenden Lochkarten soll ein Satz kreiert werden. Die zu erstellende Datei soll auf ein Magnetband geschrieben werden. Die beiden Eingabekarten, aus denen ein Satz zusammengestellt werden soll, weisen folgenden Aufbau auf:

Erstkarte

Spalten 1 bis 4:	Kundennummer
Spalte 5 :	Preisgruppe
Spalten 6 bis 40:	Kundenname
Spalten 41 bis 75:	Adresse des Kunden (1. Zeile)
Spalten 76 bis 80:	leer

Zweitkarte

Spalten 1 bis 5:	leer
Spalten 6 bis 40:	Adresse des Kunden (2. Zeile)
Spalten 41 bis 75:	Adresse des Kunden (3. Zeile)
Spalten 76 bis 80:	leer

Der in die zu erstellende Datei einzuspeichernde Satz soll die folgende Struktur besitzen:

Stelle 1 :	leer
Stelle 2 :	Preisgruppe
Stellen 3 bis 6:	Kundennummer
Stellen 7 bis 41:	Kundenname
Stellen 42 bis 76:	Adresse des Kunden (1. Zeile)
Stellen 77 bis 111:	Adresse des Kunden (2. Zeile)
Stellen 112 bis 146:	Adresse des Kunden (3. Zeile)
Stellen 147 bis 152:	Eröffnungsdatum des Kundenkontos (hier in der Form 1.3.74)
Stellen 153 bis 159:	Saldovortrag in der Form $zzz.zz (hierbei bedeuten die z Ziffern)
Stelle 160 :	leer

Der Saldovortrag soll bei der Erstellung der Datenmenge generell auf $0 gesetzt werden.

Das vorliegende Lösungsprogramm ist speziell für das Betriebssystem DOS erstellt. Beim Betriebssystem OS können die ENVIRONMENT-Attribute für die beiden Dateien EIN bzw. AUS weggelassen werden [1], im DOS sind sie bekanntlich wegen der MEDIUM-Angabe unbedingt erforderlich.

Für die Kundennummer KNR haben wir die Abbildung '9999' spezifiziert, um sicherzustellen, daß in den entsprechenden Lochkartenspalten nur Dezimalziffern abgelocht sind. Jeder Versuch, dieses Feld mit anderen Zeichen als Dezimalziffern zu versehen, führt zu unvorhersehbaren Resultaten. Bei fast allen kommerziellen Anwendungen sollte der Programmierer gezwungen werden, derart kritische Daten sorgfältig einer Plausibilitätsprüfung zu unterziehen. Mit anderen Worten, die eingegebenen Daten sollen so gewissenhaft wie möglich geprüft werden, um die Daten zurückweisen zu können, die solchen Prüfungen nicht standhalten. Nur dadurch kann die Zusammenstellung falscher Sätze vermieden werden. Wir haben hier nur die Anfänge solcher Plausibilitäts-

1) Die Kompilierer im OS beachten im übrigen die MEDIUM-Angabe nicht.

```
DERST:  PROCEDURE OPTIONS(MAIN) ;
        /* PROGRAMM ZUR ERSTELLUNG EINER DATEI MIT INFORMATIONEN UEBER */
        /* DIE USA-KUNDEN EINER HANDELSFIRMA                           */
        /*                                                             */
        /* PROGRAMMIERER:   HORST ELMAR - 10.2.1974                    */
        DECLARE  EIN FILE RECORD SEQUENTIAL BUFFERED INPUT
                     ENVIRONMENT(MEDIUM(SYS008,2540)) ;
        DECLARE  AUS FILE RECORD SEQUENTIAL BUFFERED OUTPUT
                     ENVIRONMENT(MEDIUM(SYS009,2400)) ;
        DCL  1 EGEB1,   /* EINGABEGEBIET FUER DIE ERSTKARTE */
               2 KNR PIC'9999',    /* KUNDENNUMMER */
               2 PGR CHAR(1),      /* PREISGRUPPE */
               2 KNA CHAR(35),     /* KUNDENNAME */
               2 KA1 CHAR(35),     /* KUNDENADRESSE (1. ZEILE) */
               2 LZ1 CHAR(5) ;     /* UNBENUTZTE SPALTEN */
        DCL  1 EGEB2,   /* EINGABEGEBIET FUER DIE ZWEITKARTE */
               2 LZ2 CHAR(5),      /* UNBENUTZTE SPALTEN */
               2 KA2 CHAR(35),     /* KUNDENADRESSE (2. ZEILE) */
               2 KA3 CHAR(35),     /* KUNDENADRESSE (3. ZEILE) */
               2 LZ3 CHAR(5) ;     /* UNBENUTZTE SPALTEN */
        DCL  1 AGEB,    /* AUSGABEGEBIET */
               2 LS1 CHAR(1) INITIAL(' '),
               2 PGR CHAR(1),      /* PREISGRUPPE */
               2 KNR PIC'9999',    /* KUNDENNUMMER */
               2 KNA CHAR(35),     /* KUNDENNAME */
               2 KA1 CHAR(35),     /* KUNDENADRESSE (1. ZEILE) */
               2 KA2 CHAR(35),     /* KUNDENADRESSE (2. ZEILE) */
               2 KA3 CHAR(35),     /* KUNDENADRESSE (3. ZEILE) */
               2 DTM CHAR(6),      /* ERSTELLUNGSDATUM */
               2 SLD PIC'$999V.99',
                                   /* KONTENSTAND (SALDO) */
               2 LS2 CHAR(1) INITIAL(' ') ;
        DCL  SCH BIN FIXED(15) ;   /* SCHALTER FUER DATENKARTEN */
        DCL  Z BIN FIXED(31) ;     /* ZAEHLER FUER DIE DATENSAETZE */
        ON ENDFILE(EIN)   GOTO SLU ;
        OPEN FILE(EIN), FILE(AUS) ;
        Z,SCH = 0 ;
        DTM = '1.3.74' ;
        SLD = 0.00 ;
SCHLF:  READ FILE(EIN) INTO(EGEB1) ;     /* LESEN DER ERSTKARTE */
        AGEB = EGEB1, BY NAME ;
        SCH = 1 ;
        READ FILE(EIN) INTO(EGEB2) ;     /* LESEN DER ZWEITKARTE */
        AGEB = EGEB2, BY NAME ;
        SCH = 2 ;
        WRITE FILE(AUS) FROM(AGEB) ;
        Z = Z + 1 ;
        GOTO SCHLF ;
 SLU:   /* PRUEFEN AUF PAARIGKEIT DER DATENKARTEN */
        IF  SCH=1  THEN PUT EDIT('DIE DATENKARTEN LAGEN NICHT '
                                 || 'PAARIG VOR ---> DER LETZTE'
                                 || ' SATZ WURDE NICHT AUFGENOMMEN')
                                (SKIP(2),A) ;
        PUT EDIT('ES WURDEN',Z,' SAETZE IN DIE DATEI ''AUS'' '
                 || 'AUFGENOMMEN') (PAGE,A,F(6),A) ;
      END DERST ;
```

Abb. 12.23. Beispielprogramm für die Erstellung einer Datei mittels der satzweisen EA

prüfungen gezeigt: wir haben hier nur sichergestellt, daß die letzte vom Programm eingelesene Karte eine Zweitkarte ist und das auch nur in einer ziemlich primitiven Weise.

Die ON-Anweisung mit der Bedingung

```
ENDFILE(EIN)
```

müssen wir ins Programm aufnehmen, wenn wir vermeiden wollen, daß beim Auftreten der Bedingung die Standardsystemmaßnahme ergriffen wird. Wir wollen aber zum Schluß noch die Gesamtzahl der in die Datei EIN aufgenommenen Sätze protokollieren.

Auf CLOSE-Anweisungen für die im Programm benutzten Dateien haben wir bewußt verzichtet, da es hier genügt, wenn alle Dateien implizit, d.h. automatisch, am Programmende abgeschlossen werden.

Aufgabengruppe 31

1) Schreibe das Programm zur Lösung der Aufgabe 5) der Aufgabengruppe 30 neu. Berücksichtige dabei, daß es dabei das Einlesen der Elemente jeder Matrix zuläßt, deren Zeilenzahl und deren Spaltenzahl kleiner als 11 ist. Bevor die erste zur Matrix gehörende Datenkarte eingelesen wird, ist eine Leitkarte einzulesen, die auf den Spalten 1 und die Zeilenzahl und auf den Spalten 3 und 4 die Spaltenzahl der Matrix enthält.

2) Das Programm von Aufgabe 1) ist zu modifizieren. Die Leitkarte mit der Zeilen- und Spaltenzahl der Matrix ist aus dem Datenkartenpaket zu eliminieren. Das Programm soll die Datenkarten bis zum Dateiende lesen und als Zeilenzahl bzw. Spaltenzahl der Matrix diejenigen Werte ansehen, die während des Lesens der Datenkarten als höchste Indizes festgestellt wurden.

3) Gegeben ist

```
DECLARE  J1   PIC'-V.(8)9ES99',
         J2   PIC'-9V.(7)9KS99',
         J3   PIC'S9V.(7)9E9R',
         J4   PIC'(4)9V.(4)9-',
         J5   PIC'(4)ZV.(4)9-',
         J6   PIC'ZZ,9V99.99',
         J7   PIC'Z,(3)Z,ZZ9.',
         J8   PIC'(4)-.99',
         J9   PIC'(3)Z9V.(4)9-',
         J1Ø  PIC'ZB(3)ZV.(4)9-',
         J11  PIC'*.***,****+',
         J12  PIC'*.***V,(4)*-',
         J13  PIC'999YY99',
         J14  PIC'+,+++,+V++' ;
J1,J2,J3,J4,J5,J6,J7,J8,J9,J1Ø,J11,J12,J13,J14 = 234.56 ;
```

Gesucht ist der Inhalt der den Variablen J1 bis J14 zugewiesenen Felder nach Durchführung der Ergibtanweisung!

4) Es liegt die gleiche Aufgabenstellung wie bei Aufgabe 3) vor; die den Variablen J1 bis J14 durch die Ergibtanweisung zugewiesenen Werte sind jedoch

a) -78Ø9.3724 b) -Ø.37 c) 2ØØ3 d) 7.894436

5) Auf Lochkarten sind in den Spalten 1 bis 4 ganze Zahlen mit römischen Ziffern abgelocht. Dabei sind nur die Zahlzeichen I, V, X und L benutzt worden. Schreibe ein Programm, das derartige Lochkarten liest und das nicht nur die Eingabewerte, sondern auch deren äquivalente Dezimalzahlen druckt. Dabei ist die satzweise EA zu benutzen.

6) Das Problem von Aufgabe 5) ist wie folgt abgeändert: Die römischen Zahlen belegen die Spalten 1 bis 8 der Lochkarten, wobei Leerzeichen rechts und links der eigentlichen Zahl vorhanden sein dürfen. Außerdem sind noch die Zahlzeichen C, D und M zulässig.

7) Es liegt folgender Programmabschnitt vor:

```
DCL   BH BIT(16) INIT('11Ø1ØØ1111Ø1ØØØ'B),
      CH CHAR(2) ;

CH = UNSPEC(BH) ;
PUT LIST(BH,CH) ;
```

Stelle diesen Programmabschnitt auf die satzweise Ausgabe um!

12.16. Zeigerdaten

Wir wollen ein Programm verfolgen, das zwei verschieden unterteilte Lochkarten liest und anschließend einen kurzen Satz in eine konsekutiv organisierte Datenmenge schreibt (Ausgabe auf ein Magnetband).

Die beiden zur Eingabe gelangenden Kartenarten sind wie folgt aufgebaut:

a) Kartenart 1

Spalte 1 :	1
Spalten 2 bis 9:	Datum in der Form mm/tt/jj (Monat, Tag, Jahr)
Spalten 10 bis 14:	Rechnungsnummer
Spalten 15 bis 27:	Betrag in der Form $x.xxx.xxx,xx
Spalten 28 bis 80:	leer

b) Kartenart 2

Spalte 1 :	≠1
Spalten 2 bis 13:	leer
Spalten 14 bis 21:	Datum in der Form mm/tt/jj
Spalten 22 bis 26:	Rechnungsnummer
Spalten 27 bis 39:	Betrag in der Form $x.xxx.xxx,xx
Spalten 40 bis 80:	leer

Beide Kartenarten weisen also im Grunde genommen den gleichen Inhalt auf, nur daß die Datenfelder auf den Karten anders angeordnet sind. Innerhalb des Eingabestroms treten die beiden Kartenarten miteinander vermischt auf. Somit kann der Programmierer nicht im voraus bestimmen, welche Kartenart als nächste zu erwarten ist, und er weiß infolgedessen auch nicht, in welche Struktur er den Karteninhalt einzulesen hat und wie er ihn verarbeiten soll.

Aus beiden Kartenarten soll der gleiche Ausgabesatz zusammengestellt werden. Seine Einteilung ist wie folgt vorgesehen:

Stelle 1 :	Kartenart
Stellen 2 bis 6:	Rechnungsnummer
Stellen 7 bis 11:	Rechnungsbetrag (Dezimalzahl in Festpunktdarstellung)

Stellen 12 bis 19: Datum in der Form mm/tt/jj
Stellen 20 bis 24: leer

Wegen des verschiedenen Aufbaus der Lochkarten ist die Anwendung der reihenweisen EA nicht möglich. Eine Lösungsmöglichkeit bietet sich an, wenn wir die Karten in eine gegenüber den Kartenarten neutrale Struktur lesen. Danach ermitteln wir die Kartenart. Unter Belassung der Daten auf dem Platz, auf den sie eingelesen wurden, setzen wir einfach einen Zeiger auf den Anfang des Teiles des Eingabegebietes, das sich durch die unterschiedliche Zusammenstellung der Daten auszeichnet. Diese Lösungsmöglichkeit ist in der Abb. 12.24. dargestellt.

```
VKB:  PROCEDURE OPTIONS(MAIN) ;

      /*** VERARBEITUNG VERSCHIEDENER KARTENARTEN ***/
      /* PROGRAMMIERT VON HORST ELMAR --- 10.2.1974 */
      /* ====        (BETRIEBSSYSTEM OS)          ==== */

      DECLARE  EIN FILE RECORD BUFFERED SEQUENTIAL INPUT,
               AUS FILE RECORD BUFFERED SEQUENTIAL OUTPUT ;

      DCL   1 EGEB,    /* EINGABEVARIABLE FUER BEIDE KARTENARTEN */
                2 ART CHAR(1),        /* KARTENART */
                2 REST CHAR(79) ;     /* REST */
      DCL   1 KA1 BASED(ZE),          /* KARTENART 1 */
                2 DTM PIC'99/99/99',          /* DATUM */
                2 RNR PIC'99999',             /* RECHNUNGSNUMMER */
                2 BTR PIC'$Z.ZZZ.ZZ9V,99',    /* RECHNUNGSBETRAG */
                2 LEZ CHAR(53) ;
      DCL   1 KA2 BASED(ZE),          /* KARTENART 2 */
                2 LZ1 CHAR(12),
                2 DTM PIC'99/99/99',          /* DATUM */
                2 RNR PIC'99999',             /* RECHNUNGSNUMMER */
                2 BTR PIC'$Z.ZZZ.ZZ9V,99',    /* RECHNUNGSBETRAG */
                2 LZ2 CHAR(41) ;
      DCL   1 AGEB BASED(ZA),    /* AUSGABESTRUKTUR */
                2 KA CHAR(1),                 /* KARTENART */
                2 RNR PIC'(5)9',              /* RECHNUNGSNUMMER */
                2 BTR DECIMAL FIXED(9,2),     /* RECHNUNGSBETRAG */
                2 DTM PIC'99/99/99',          /* DATUM */
                2 RST CHAR(5) ;               /* UNBENUTZTE STELLEN */
      DCL   (ZE,ZA) POINTER ;

      ON ENDFILE(EIN)  GOTO E ;    /* DATEIENDE-BEDINGUNG */

 SL:  /* SCHLEIFE */
      READ FILE(EIN) SET(ZE) ;
      LOCATE AGEB FILE(AUS) SET(ZA) ;
      IF  ART='1'  THEN  AGEB = KA1, BY NAME ;
                  ELSE  AGEB = KA2, BY NAME ;
      KA = ART ;
      GOTO SL ;

  E:  DISPLAY ('PROGRAMM  ''VKB'' BEENDET') ;
    END VKB ;
```

Abb. 12.24. Beispielprogramm zur Verarbeitung verschiedener Kartenarten

Das Strukturelement ART nimmt nach dem Lesen einer Datenkarte das die Kartenart kennzeichnende Zeichen auf, die restlichen Zeichen der Lochkarte werden als 79stellige Zeichenkette aufgefaßt und nach dem Strukturelement REST gestellt. Die Struktur EGEB umfaßt somit gerade 80 Zeichen.

Die unterschiedliche Anordnung der Daten auf den Lochkarten bei den verschiedenen Kartenarten hat uns veranlaßt, zwei Strukturen, nämlich KA1 und KA2, zu definieren, die die Einteilung der Spalten 2 bis 80 bei den beiden Kartenarten widerspiegeln. Das Attribut BASED, das wir den beiden Hauptstrukturnamen beigestellt haben, besagt jedoch, daß die betreffende Struktur, auf eine Basis bezogen werden soll, d.h. keinen eigenen Hauptspeicherplatz beanspruchen wird. Die Beziehung zur Basisvariablen wird durch einen Zeiger hergestellt. Eine Variable, die wie KA1 und KA2 nur über einen zur Basis verweisenden Zeiger angesprochen (und damit adressiert) werden kann, bezeichnet man als basisbezogene Variable. Da KA1 und KA2 über die gleiche Zeigervariable ZE auf die Basis bezogen werden sollen, ergeben sich folgerichtig zwei Beschreibungen für die gleiche Basisvariable. Um ein Gleichnis zu gebrauchen, es ist gewissermaßen so, als ob wir zwei Schablonen in die Hand bekommen hätten, die wir auf die gleiche Unterlage legen könnten, wobei uns durch einen Zeiger die Unterlage bestimmt wird. Verwenden wir nach erfolgter Überlagerung die Namen von KA1, so können wir die Felder ansprechen, die bei der Kartenart 1 vorhanden sind. Verwenden wir hingegen die Namen von KA2, so können wir die der Kartenart 2 entsprechenden Felder ansprechen.

Der Zeigervariablen ZE weisen wir durch die eingefügte Funktion ADDR den Zeigerwert zu, der auf die Basisvariable verweist. In unserem Programmbeispiel finden wir infolgedessen die Ergibtanweisung

```
ZE = ADDR(REST) ;
```

Durch diese Anweisung haben wir REST als Basisvariable eingeführt.

ADDR ist eine spezielle eingefügte Funktion. Sie liefert als Funktionswert einen Zeigerwert ab, durch den der Hauptspeicherplatz erkannt wird, der dem Argument zugeordnet ist (mit anderen Worten: die Adresse des Hauptspeicherplatzes). Nachdem ZE mit diesem Wert versorgt ist, liegt auch eine konkrete Basis vor, so daß nunmehr die beiden Strukturen KA1 und KA2 Bedeutung erlangt haben.

Die bloße Anwesenheit des Namens ZE in Klammern hinter dem Attribut BASED genügt vollauf, um ZE als Zeigervariable einzuführen. Es ist somit nicht notwendig, ZE in einer gesonderten Vereinbarung als Zeigervariable zu erklären. Wir müssen eigentlich eine solche Erklärung nur dann ins Programm aufnehmen, wenn wir ZE zusätzliche Attribute verpassen wollen. Dennoch haben wir ZE explizit im Programm als Zeigervariable mit Hilfe des hierfür geschaffenen Attributes POINTER festgelegt; es ging uns dabei lediglich darum, eine solche Vereinbarung einmal zu demonstrieren. Zeigervariable können selbstverständlich auch Elemente von Bereichen und Strukturen sein.

Zeigervariable können wir in Ergibtanweisungen verwenden, doch müssen derartige Ergibtanweisungen in der Form

```
zeigervariable = zeigerausdruck ;
```

vorliegen. Unter einem Zeigerausdruck verstehen wir entweder eine Zeigervariable oder eine Funktion, deren Funktionswert vom Zeigertyp ist. Als Beispiel soll

```
Z = Y ;
```

gegeben werden. Durch diese Ergibtanweisung wird die Zeigervariable Z auf den augenblicklichen Wert der Zeigervariablen Y gesetzt. Zwei Zeigerfunktionen sind in PL/1 eingefügt:

```
ADDR(variable)
NULL
```

Durch ADDR wird ein Zeiger auf den Wert gesetzt, der gleich der Hauptspeicheradresse der spezifizierten Variablen ist, die eine Strukturvariable sein kann, aber nicht zu sein braucht. Die eingefügte Funktion NULL setzt die Zeigervariable bzw. die Zeigervariablen auf der linken Seite einer Ergibtanweisung auf einen speziellen Nullwert, durch den keine Hauptspeicheradresse mehr identifiziert wird. Diese beiden Funktionen sind in erster Linie bei der sogenannten Listenverarbeitung, einer speziellen Programmierungstechnik, von besonderem Nutzen.

Zeigervariable können auch in Vergleichsausdrücke eingehen; jedoch sind hier nur die beiden Operatoren = und ¬= zulässig. Zum Beispiel ermöglicht die IF-Anweisung

```
IF  Z¬=NULL  THEN ···
```

die Überprüfung des Wertes der Zeigervariablen Z: Zeigt sie noch auf einen bestimmten Hauptspeicherplatz (THEN-Klausel) oder nicht.

Durch die satzweise EA können Zeigerwerte sowohl geschrieben als auch gelesen werden, doch ist hierbei eine gewisse Vorsicht geboten. Wenn Zeigerwerte nicht im gleichen Jobstep von derselben Prozedur gelesen werden, in der sie geschrieben wurden, so werden sie im allgemeinen nicht die gleichen Daten identifizieren. Somit ist es kaum sinnvoll, Zeigerwerte zu schreiben und zu lesen. Bei der reihenweisen EA können Zeigerwerte weder gelesen noch geschrieben werden.

Für basisbezogene Variable dürfen weder das Attribut INITIAL noch das Attribut EXTERNAL benutzt werden. Die Zeigervariable, die zur Kenntlichmachung einer basisbezogenen Variablen herangezogen werden soll, darf nicht indiziert sein. Weiterhin darf sie nicht als Element irgendeiner basisbezogenen Struktur erscheinen.

Die Deklaration einer basisbezogenen Variablen bewirkt keine Hauptspeicherplatzreservierung; wir sagten das schon einmal. Tatsächlich wird durch sie nur eine Schablone festgelegt, die über diejenigen Hauptspeicherplätze gelegt wird, auf die der zugehörige Zeiger verweist. Bei jeder Referenz auf ein Element einer basisbezogenen Variablen ist gleichzeitig eine Referenz auf den augenblicklichen Wert der zugehörigen Zeigervariablen eingeschlossen. Soll jedoch eine andere Zeigervariable herangezogen werden, um die basisbezogene Variable einer anderen Basis aufzuprägen, so können wir beispielsweise wie folgt codieren:

```
ZE = Q ;
AGEB.DTM = KA2.DTM ;
```

Wir haben uns hierbei auf das Programmbeispiel in der Abb. 12.24. bezogen und eine Datumszuweisung vorgenommen. Q sei eine Zeigervariable, die mit

```
DCL  Q POINTER ;
```

erklärt ist und der irgendein Zeigerwert zugewiesen sein soll. Diese Codierung weist ein entscheidendes Handicap auf: Sie zerstört den bisherigen Wert von ZE. Unter Benutzung des Zeiger-Kennzeichnungssymboles -> (Minuszeichen gefolgt vom „Größer-als"-Zeichen) können wir die obige Statementfolge ersetzen durch

```
AGEB.DTM = Q -> KA2.DTM ;
```

Durch diese Codierung bleibt uns der Wert von ZE erhalten. Dem Kompilierer sagt diese Schreibweise aus, daß er KA2 derjenigen Basis aufprägen soll, die durch den Zeiger Q an Stelle des regulären Zeigers ZE identifiziert wird.

12.17. Die satzweise EA unter Benutzung von Puffern

Wenn wir die READ-Anweisung mit der INTO-Option benutzen, so erzeugt der Kompilierer Maschineninstrunktionen, die die Daten zunächst in einen Puffer lesen und dann sie von dort in die Struktur übertragen, die durch die INTO-Option angesprochen ist. Bei der Ausgabe läuft ein analoger Vorgang ab: Die Daten werden zunächst von der Struktur, die in der FROM-Option der WRITE-Anweisung aufgeführt ist, in einen Puffer übertragen und von dort erst werden sie ausgegeben. Puffer sind nichts anderes als Hauptspeichergebiete, in denen Daten zeitweise verweilen. Mitunter wird gewünscht, daß die Zeit eingespart werden soll, die durch die Datenübertragungen im Hauptspeicher zwischen Puffern und Strukturen vergeudet wird.

Für die Eingabe kann hierzu das READ-Statement in der Form

```
READ FILE(dateiname) SET(zeigervariable) ;
```

benutzt werden. In diesem Fall wird unmittelbar nach der Zurverfügungstellung eines Satzes die Zeigervariable auf einen Wert gesetzt, durch den der Beginn des gelesenen Satzes identifiziert werden kann. Nunmehr kann eine beliebig basisbezogene Variable, für die die betreffende Zeigervariable erklärt ist, zur Verarbeitung herangezogen werden.

Die Ausgabesituation ist ein wenig diffiziler. Eingesetzt werden muß das Statement

```
LOCATE variable FILE(dateiname) SET(zeigervariable) ;
```

Unter variable ist hierbei eine nichtindizierte basisbezogene Variable mit der Stufennummer 1 zu verstehen. Durch das LOCATE-Statement wird im Puffer ein Teilgebiet reserviert, dessen Länge gerade ausreicht, um die basisbezogene Variable darauf legen zu können. Außerdem wird der Zeigervariablen ein Wert zugewiesen, der das bereitgestellte Pufferteilgebiet identifiziert. Das Hineinstellen von Daten in dieses Pufferteilgebiet erfolgt jedoch nicht und damit auch keine sich unmittelbar anschließende Ausgabe. Um es ganz klar auszudrücken, durch die LOCATE-Anweisung wird dem Benutzer nur ein Gebiet zur Verfügung gestellt, in dem er seinen Ausgabesatz selbst zusammenbauen muß. Die eigentliche Ausgabe geschieht während der Ausführung der nächsten LOCATE-Anweisung, die sich auf die gleiche Datei bezieht, oder bei der CLOSE-Anweisung für die betreffende Datei oder, falls diese fehlt, bei Abschluß des Jobs.

In der Abb. 12.25. finden wir ein Programm aufgelistet, das derselben Aufgabenstellung gerecht wird, wie wir sie im Abschnitt 12.16. kennengelernt haben und für die eine erste Lösungsversion in der Abb. 12.24. vorgestellt wurde. Bei der Lösung von Abb. 12.25. wird sowohl bei der Eingabe als auch bei der Ausgabe mit Puffern gearbeitet, d.h. die Daten werden von den Eingabepuffern unmittelbar in die Ausgabepuffer übertragen. Die Eingabepuffer werden dabei von der Zeigervariablen ZE, die Ausgabepuffer von der Zeigervariablen ZA regiert. In vielen Fällen wird man in der Praxis dazu neigen, beide Modi der EA-Anweisungen im gleichen Programm einzusetzen.

Die SET-Option kann beim LOCATE-Statement entfallen. In diesem Fall nimmt der Kompilierer standardmäßig an, daß die Zeigervariable benutzt werden soll, die in der Erklärung der basisbezogenen Variablen variable aufgeführt ist.

Voraussetzung für die Anwendung der Verarbeitung von Daten in Puffern ist die sequentielle Zugriffsmethode für die spezifizierte Datei, wobei ebenso selbstverständlich ist, daß für sie außerdem das Attribut BUFFERED vorliegt.

```
VKA:  PROCEDURE OPTIONS(MAIN) ;

      /*** VERARBEITUNG VERSCHIEDENER KARTENARTEN ***/
      /* PROGRAMMIERT VON HORST ELMAR --- 10.2.1974 */
      /* ====        (BETRIEBSSYSTEM OS)          ==== */

      DECLARE  EIN FILE RECORD BUFFERED SEQUENTIAL INPUT,
               AUS FILE RECORD BUFFERED SEQUENTIAL OUTPUT :

      DCL  1 EGEB,    /* EINGABEVARIABLE FUER BEIDE KARTENARTEN */
             2 ART CHAR(1),        /* KARTENART */
             2 REST CHAR(79) ;     /* REST */
      DCL  1 KA1 BASED(ZE),        /* KARTENART 1 */
             2 DTM PIC'99/99/99',          /* DATUM */
             2 RNR PIC'99999',             /* RECHNUNGSNUMMER */
             2 BTR PIC'$Z.ZZZ.ZZ9V,99',    /* RECHNUNGSBETRAG */
             2 LEZ CHAR(53) ;
      DCL  1 KA2 BASED(ZE),        /* KARTENART 2 */
             2 LZ1 CHAR(12),
             2 DTM PIC'99/99/99',          /* DATUM */
             2 RNR PIC'99999',             /* RECHNUNGSNUMMER */
             2 BTR PIC'$Z.ZZZ.ZZ9V,99',    /* RECHNUNGSBETRAG */
             2 LZ2 CHAR(41) ;
      DCL  1 AGEB,    /* AUSGABEVARIABLE */
             2 KA CHAR(1),                 /* KARTENART */
             2 RNR PIC'(5)9',              /* RECHNUNGSNUMMER */
             2 BTR DECIMAL FIXED(9,2),     /* RECHNUNGSBETRAG */
             2 DTM PIC'99/99/99',          /* DATUM */
             2 RST CHAR(5) ;               /* UNBENUTZTE STELLEN */
      DCL  ZE POINTER ;

      ON ENDFILE(EIN)  GOTO E ;    /* DATEIENDE-BEDINGUNG */

      RST = (5)' ' ;
      ZE = ADDR(REST) ;
SL:   /* SCHLEIFE */
      READ FILE(EIN) INTO(EGEB) ;
      IF  ART='1'  THEN  AGEB = KA1, BY NAME ;    /* KARTENART '1' */
                 ELSE  AGEB = KA2, BY NAME ;    /* KARTENART '2' */
      KA = ART ;
      WRITE FILE(AUS) FROM(AGEB) ;
      GOTO SL ;

 E:   DISPLAY ('PROGRAMM  ''VKA'' BEENDET') ;
    END VKA ;
```

Abb. 12.25. Satzweise EA mit Puffern

12.18. Die GET- und PUT-Anweisungen mit der Option STRING

Das in Abschnitt 12.16. vorgestellte Problem (Abb. 12.24. und 12.25.) kann noch auf einem anderen Weg gelöst werden. Das zur Demonstration dieser dritten Lösungsmethode geschriebene Programm ist in der Abb. 12.26. aufgelistet.

```
VKC:  PROCEDURE OPTIONS(MAIN) ;

      /*** VERARBEITUNG VERSCHIEDENER KARTENARTEN ***/
      /* PROGRAMMIERT VON HORST ELMAR --- 10.2.1974 */
      /* ====        (BETRIEBSSYSTEM OS)          ==== */

      DECLARE  EIN FILE RECORD BUFFERED SEQUENTIAL INPUT,
               AUS FILE RECORD BUFFERED SEQUENTIAL OUTPUT ;

      DCL  1 EGEB,    /* EINGABEVARIABLE FUER BEIDE KARTENARTEN */
             2 ART CHAR(1),       /* KARTENART */
             2 REST CHAR(79) ;    /* REST */
      DCL  1 AGEB,    /* AUSGABEVARIABLE */
             2 KA CHAR(1),                  /* KARTENART */
             2 RNR PIC'(5)9',               /* RECHNUNGSNUMMER */
             2 BTR DECIMAL FIXED(9,2),      /* RECHNUNGSBETRAG */
             2 DTM PIC'99/99/99',           /* DATUM */
             2 RST CHAR(5) INIT((5)' ') ;

      ON ENDFILE(EIN)  GOTO E ;    /* DATEIENDE-BEDINGUNG */

SL:   /* SCHLEIFE */
      READ FILE(EIN) INTO(EGEB) ;
      IF  ART='1'  THEN
        GET STRING(REST) EDIT(DTM,RNR,BTR)
                         (P'99/99/99',P'(5)9',P'Z.ZZZ.ZZ9V,99')
                 ELSE
        GET STRING(REST) EDIT(DTM,RNR,BTR)
                         (X(12),P'99/99/99',P'(5)9',
                          P'Z.ZZZ.ZZ9V,99') ;
      WRITE FILE(AUS) FROM(AGEB) ;
      GOTO SL ;

 E:   DISPLAY ('PROGRAMM  ''VKC'' BEENDET') ;
      END VKC ;
```

Abb. 12.26.

In diesem Programm begegnet uns eine neue Version des bereits bekannten GET-Statements, nämlich

GET STRING(zeichenkettenvariable) EDIT(datenliste) (formatliste) ;

In jedem GET- oder PUT-Statement können wir an Stelle von FILE(dateiname) auch STRING(zeichenkettenvariable) schreiben; ein solches Statement wird genau so wie jedes reguläre GET/PUT-Statement ausgeführt, nur daß nicht zu einer externen Datenmenge zugegriffen wird, sondern zu einer Zeichenkettenvariablen. In unserem Beispiel ist es die Variable REST. Somit ist die Anweisung

GET STRING(REST) EDIT(···) (···) ;

in vieler Hinsicht äquivalent der formatgesteuerten Eingabe von Daten von einer externen Datei, die die Zeichenkette enthält, die jetzt der Variablen REST zugewiesen ist. Als Resultat der beiden GET-Anweisungen werden der Variablen AGEB die fehlenden Werte zugewiesen. Die Zeichenkettenvariable REST bleibt dabei wertmäßig unverändert.

Die in den Formatlisten der formatgesteuerten GET-Anweisungen enthaltenen P-Formatelemente entsprechen vollkommen dem PICTURE-Attribut. Die Abbildungen werden in gleicher Weise aus Abbildungszeichen zusammengestellt. Die Abbildungszeichen selbst sind ebenfalls

bedeutungsgleich. Der einzige Unterschied zwischen Attribut und Formatelement besteht in der Tatsache, daß durch das PICTURE-Attribut die Zeichenfolge von Variablen im Hauptspeicher beschrieben wird, während P-Formatelemente die Zeichenfolge von Daten auf externe Speichermedien (Datenträger) oder von Zeichenkettenvariablen beschreiben, je nachdem, ob die EA-Anweisungen die FILE-Option oder die STRING-Option enthalten.

Wenn die Zeichenkettenvariable in der STRING-Option nicht lang genug ist, um die von der Datenliste und der Formatliste insgesamt angeforderten Zeichen abliefern zu können, so tritt die ERROR-Bedingung auf. Zu dieser kommt es auch, wenn die Zeichenkettenvariable nicht definiert ist.

Die Optionen FILE, SKIP, PAGE, LINE und COPY können nicht zusammen mit der STRING-Option benutzt werden. Jede GET-Anweisung mit der STRING-Option greift immer zum Anfang der spezifizierten Zeichenkettenvariablen zu. In alle Formen der reihenweisen EA (daten-, listen- und formatgesteuert) darf die STRING-Option eingehen, aber sie ist selten sinnvoll anzuwenden mit Ausnahme der formatgesteuerten reihenweisen EA.

12.19. Die Pseudovariablen

Die folgenden Funktionen dürfen in Ergibtanweisungen auf der linken Seite des Ergibtzeichens verwendet werden; ebenso dürfen sie als Laufvariable in DO-Anweisungen und als Datenelement in Datenlisten bei der GET-Anweisung auftreten. Auf der rechten Seite von Ergibtzeichen dürfen sie wie jede andere Funktion verwendet werden. Wenn die eingefügten Funktionen in der soeben beschriebenen Weise benutzt werden, heißen sie Pseudovariable. Sie werden genau so geschrieben wie die ihnen entsprechenden eingefügten Funktionen. Eine Ausnahme ist jedoch zu beachten: Die Argumente von Pseudovariablen dürfen selbst keine eingefügten Funktionen enthalten, die Pseudovariablen entsprechen. Somit ist z.B.

```
UNSPEC(SUBSTR(KETTE,4,2)) = X ;
```

ein ungültiges Statement.

Nachfolgend wollen wir einige Pseudovariablen besprechen:

a) SUBSTR(k,i [,j])
Unter k ist eine Kettenvariable (Zeichenkette oder Bitkette) zu verstehen. Die beiden weiteren Argumente i und, falls vorhanden, j stellen Ausdrücke dar, die in ganze Zahlen umgewandelt werden. (Sollte k keine Kettenvariable sein, so wird k in eine solche umgewandelt.) Der auf der rechten Seite des Ergibtzeichens stehende Ausdruck wird in eine Kette des Typs von k umgewandelt und dann derjenigen Teilkette von k zugewiesen, die an der i-ten Stelle von k beginnt und j Stellen umfaßt. Fehlt das Argument j, so läuft die zeichen- bzw. bitweise Zuweisung bis zum Ende von k, d.h. j wird gleich n-i+1 gesetzt, wobei n gleich der Länge von k ist. Betrachten wir ein einfaches Beispiel (Abb. 12.27.).

b) STRING(x)
Unter x ist ein Bereich oder eine Struktur zu verstehen, dessen bzw. deren Elemente sämtlich Zeichenketten oder numerische Daten in Zeichenform (PICTURE-Attribut) darstellen. Durch die eingefügte Funktion STRING wird bewirkt, daß alle Elemente von x in der Reihenfolge ihrer Anordnung im Hauptspeicher zu einer einzigen Zeichenkette zusammengeschlossen werden. Wird STRING als Pseudovariable eingesetzt, so wird die auf der rechten Seite des Ergibtzeichens sich ergebende Zeichenkette den verketteten Elementen von x zugewiesen.

```
DCL ZK1 CHAR(8) INITIAL('MARIANNE') ;
DCL ZK2 CHAR(7) INITIAL('PAULINE') ;
DCL ZK3 CHAR(6) INITIAL('123456') ;

ZK3 = SUBSTR(ZK2,2,4) ;
SUBSTR(ZK1,4,2) = SUBSTR(ZK2,6,2) ;
```

Ergebnisse

```
ZK3 = 'AULIbb' ;
ZK1 = 'MARNENNE' ;
```

Anmerkung: b bedeutet hier das Leerzeichen.

Abb. 12.27. Eingefügte Funktion und Pseudovariable SUBSTR

c) UNSPEC(v)
Die skalare Variable v kann entweder vom arithmetischen Typ oder vom Kettentyp sein, jedoch keinesfalls vom Markentyp. Der sich auf der rechten Seite des Ergibtzeichens ergebende Wert wird in eine Bitkette umgewandelt. Dieser wird der Bitkette zugewiesen, die durch v repräsentiert wird, wobei keine Umwandlung in den Typ von v stattfindet (die Länge dieser Bitkette ist also gleich der Anzahl Bitstellen, die sich aus den von v belegten Bytes ergibt). Bleiben wir bei einem Beispiel (Abb. 12.28.).

```
DCL  A BIT(8) INT('11ØØØØØ1'B) ;
DCL  B CHARACTER(1) ;
UNSPEC(B) = A ;
```

Resultat: B weist damit das Bitmuster '11ØØØØØ1'B auf.
Dieses entspricht aber im EBCDI-Code dem
Buchstaben A, so daß B den Wert 'A' besitzt.

Abb. 12.28. Beispiel für die Pseudovariable UNSPEC

d) COMPLEX(a,b)
Der Realteil des auf der rechten Seite des Ergibtzeichens stehenden Ausdruckes wird der Variablen a zugewiesen, der Imaginärteil der Variablen b.

e) REAL(c)
Der Realteil des auf der rechten Seite des Ergibtzeichens stehenden Ausdruckes wird dem Realteil der Variablen c zugewiesen, der Imaginärteil der komplexen Variablen c bleibt unangetastet.

f) IMAG(c)
Der Imaginärteil des auf der rechten Seite des Ergibtzeichens stehenden Ausdrucks wird dem Imaginärteil der komplexen Variablen c zugewiesen, der Realteil von c bleibt unangetastet.

Die folgenden beiden Pseudovariablen dienen dazu, die Fehlerquellen zu bestimmen, die die Ursache beim Auftreten einer CONVERSION-Bedingung waren; ihre Verwendung ist also auf Oneinheiten beschränkt.

a) ONSOURCE
Der Wert des auf der rechten Seite des Ergibtzeichens stehenden Ausdruckes wird in eine Zeichenkette umgewandelt und der Variablen zugewiesen, die bei der Umwandlung das Auftreten der CONVERSION-Bedingung verursachte. Diese Zeichenkette wird im Bedarfsfall nach rechts mit Leerzeichen aufgefüllt; ihre Länge ist jedenfalls gleich der Länge der den Umwandlungsfehler verursachenden Variablen.

b) ONCHAR
Der Wert des auf der rechten Seite des Ergibtzeichens stehenden Ausdruckes wird in eine Zeichenkette mit der Länge 1 umgewandelt und der Zeichenstelle zugewiesen, die das Auftreten der CONVERSION-Bedingung verursachte.

Nach Rückkehr von der Oneinheit werden die durch ONSOURCE bzw. ONCHAR neu gesetzten Werte benutzt. Bei GET-Anweisungen der daten- bzw. listengesteuerten Form sollte nur ein fehlerhaftes Datenfeld durch ONSOURCE korrigiert werden; eingebettete Leerzeichen oder Kommas könnten bei Korrektur mehrerer Datenfelder erneut das Auftreten einer CONVERSION-Bedingung verursachen, und zwar dann, wenn sich dem Statement mit ONSOURCE eine normale Rückkehr anschließt.

Aufgabengruppe 32

1) Schreibe ein Programm, das Lochkarten liest, bei denen die Spalten 1 bis 6 ein Datum in der Form mmttjj enthalten. Bei einem gültigen Datum ist dieses in die Form jjttt umzuwandeln. Es ist das Datum in der Eingabeform und in der Ausgabeform zu drucken. Bei einem ungültigen Datum ist die Eingabeform und zusätzlich die Meldung

UNGUELTIG

zu drucken.

Bei der Eingabeform bedeuten mm den Monat (1 bis 12), tt den Tag (1 bis 28 bzw. 29 bzw. 30 bzw. 31) und jj die letzten beiden Ziffern des Jahres (00 bis 99). Bei der Ausgabeform bedeuten jj die beiden letzten Ziffern des Jahres und ttt den laufenden Tag des Jahres (1 bis 365 bzw. 366).

2) Das Programm der Aufgabe 1) ist wie folgt zu ergänzen:
Es ist außerdem der Wochentag zu ermitteln und zu drucken, wenn ein Jahr zwischen 1967 und 1978 angegeben ist.
Voraussetzung: Es sind eingabemäßig alle Jahresangaben als zum 20. Jahrhundert zugehörend zu betrachten

3) Welche Ausgaben werden durch den folgenden Programmabschnitt veranlaßt?

```
DCL   MA CHAR(2),
      RI BIT(16) INIT('11Ø1ØØ111111Ø1ØØ'B) ;
MA = UNSPEC(RI) ;
PUT SKIP(2) LIST (MA) ;
UNSPEC(MA) = RI ;
PUT SKIP LIST (MA) ;
```

4) Programmiere die Lösung des Beispiels im Abschnitt 12.9. unter Benutzung der satzweisen EA neu. Für die Datenumwandlungen sind GET/PUT-Anweisungen mit der Option STRING einzusetzen.

5) Schreibe ein Programm, das Lochkarten liest, auf denen je zwei sechsstellige Wörter auf den Spalten 1 bis 6 bzw. 7 bis 12 abgelocht sind. Keines der Wörter enthält einen Buchstaben mehrfach. Drucke die eingelesenen Wörter und die Zahl derjenigen Buchstaben, die sowohl im ersten als auch im zweiten Wort vorhanden sind.
Beispiel: Die Wörter DRUCKE und DRACHE haben vier gemeinsame Buchstaben

6) Das Programm von Aufgabe 5) ist so zu erweitern, daß vor der Ermittlung der Anzahl der gemeinsamen Buchstaben der beiden Wörter zunächst jedes Wort dahingehend untersucht wird, ob es jeden Buchstaben auch tatsächlich nur einmal enthält. Im verneinenden Fall ist eine Fehlernachricht zu drucken.

13. Fortgeschrittene Sprachelemente von PL/1

13.1. Das Aufdatieren von Datenmengen bei Benutzung der sequentiellen Zugriffsmethode

Datenmengen, deren Sätze mit Hilfe der sequentiellen Zugriffsmethode aufdatiert werden sollen, müssen wie folgt angesprochen werden:

a) Zunächst ist wie üblich der Satz mit der READ-Anweisung zu lesen.

b) Anschließend ist der geänderte Satz zurückzuschreiben, wozu die Anweisung

```
REWRITE FILE(dateiname) [FROM(variable)] ;
```

dient.

Das REWRITE-Statement wirkt im Grunde wie das WRITE-Statement; jedoch ist zu beachten, daß es den zuletzt gelesenen Satz überschreibt, d.h. ersetzt. Das WRITE-Statement hingegen veranlaßt, daß ein neuer Satz in die Datei aufgenommen wird.

Ist die FROM-Option in die REWRITE-Anweisung eingeschlossen, dann wird derjenige Satz zurückgeschrieben, der zur Variablen variable gehört. Fehlt diese Option dagegen, so wird der Satz zurückgeschrieben, der im Puffer steht.

Wenn ein Satz mit der um die INTO-Option ergänzten READ-Anweisung gelesen wurde, es der nachfolgenden REWRITE-Anweisung jedoch an der FROM-Option mangelt, so wird der gelesene Satz zurückgeschrieben; das aber bedeutet praktisch eine unnötige Operation. Somit ist zu erwarten, daß auf eine READ-Anweisung mit der INTO-Option eine REWRITE-Anweisung mit der FROM-Option folgt.

Wird ein Satz mit einer READ-Anweisung gelesen, die die SET-Option aufweist, so wird durch eine REWRITE-Anweisung ohne FROM-Option von demjenigen Puffergebiet zurückgeschrieben, in dem der gelesene Satz zur Verarbeitung bereitgestellt wurde. Auf diese Weise erfolgt in der Regel das Aufdatieren von Sätzen in Puffern.

Eine Datei mit dem Attribut UPDATE ist gegenüber einer Datei mit dem Attribut INPUT um die sehr entscheidende Fähigkeit erweitert, daß die in ihr enthaltenen Sätze nicht nur gelesen, sondern auch aufdatiert werden können.

13.2. Das Verarbeiten von Datenmengen ohne Benutzung von Puffern

Die beiden Dateiattribute BUFFERED und UNBUFFERED können sowohl für Dateien angegeben werden, die auf Magnetbändern residieren,als auch für solche, die auf Plattenstapeln liegen. Tatsächlich ist es möglich, die beiden Attribute beliebigen Dateien zuzuordnen, doch sind sie meist nur solchen Dateien sinnvoll zuzuordnen, die auf den beiden oben erwähnten Speichermedien angeordnet sind. Das Attribut UNBUFFERED bedeutet, daß die durch die EA-Operationen angesprochenen Sätze während der Eingabe bzw. Ausgabe keine Puffer passieren. Nur ein einziges EA-Gebiet wird für solche Dateien benutzt, was zur Folge hat, daß zwischen der EA und der Verarbeitung keine Überlappung erfolgen kann. In der Tat erzeugt der Kompilierer „versteckte" Puffer auch für Dateien mit dem Attribut UNBUFFERED, so daß eigentlich durch Benutzung dieses Attributes kaum etwas gewonnen wird.

Für Dateien mit dem Attribut UNBUFFERED können die folgenden Anweisungen herangezogen werden:

```
I.   READ FILE(dateiname) INTO(variable) ;
II.  WRITE FILE(dateiname) FROM(variable) ;
III. REWRITE FILE(dateiname) FORM(variable) ;
```

Nicht verwendet werden können jedoch die folgenden Formen der EA-Anweisungen:

```
I.   READ FILE(dateiname) SET(zeigervariable) ;
II.  REWRITE FILE(dateiname) ;
III. LOCATE variable FILE(dateiname) [SET(zeigervariable)] ;
```

Dateien mit dem Attribut UNBUFFERED werden relativ häufig als Arbeitsdateien entworfen, d.h. für Dateien, die mehr oder weniger nur den Zweck erfüllen, als Erweiterung des Hauptspeichers während der Programmausführung zu dienen. Deshalb wird eine solche Datei gewöhnlich zunächst mit dem Attribut OUTPUT eröffnet, dann abgeschlossen und im weiteren Programmablauf mit dem Attribut INPUT eröffnet.

Auf Dateien mit dem Attribut BUFFERED wirken stets die erweiterten sequentiellen Zugriffsmethoden ein, bei Dateien mit dem Attribut UNBUFFERED sind es stets die Basismethoden.

Eine im ENVIRONMENT-Attribut aufgeführte Angabe BUFFERS besitzt nur für Dateien mit dem Attribut BUFFERED Gültigkeit; für Dateien mit dem Attribut UNBUFFERED wird diese Angabe ignoriert.

13.3. Die Zugriffsmethoden bei Datenmengen mit regionaler Organisationsform

Die regionale Organisationsform kann nur Datenmengen unterstellt werden, die auf SEDZ [1] residieren. Wird auf sie die sequentielle Zugriffsmethode bezogen, so können die folgenden Anweisungen uneingeschränkt benutzt werden:

```
I.   READ FILE(dateiname) INTO(variable) [KEYTO(sv)] ;
II.  READ FILE(dateiname) [IGNORE(ausdruck)] ;
III. REWRITE FILE(dateiname) [FROM(variable)] ;
IV.  WRITE FILE(dateiname) FROM(variable) KEYFROM(ausdruck) ;
```

Dateien, deren Dateiname mit dem Attribut INPUT erklärt ist, dürfen nur mit READ-Statements angesprochen werden. Das Attribut OUTPUT wiederum kennzeichnet Dateien, auf die allein WRITE-Anweisungen bezogen werden dürfen. Bei Dateien mit dem Attribut UPDATE muß auf die Verwendung des WRITE-Statements verzichtet werden.

Bezüglich der Pufferungsattribute ist folgendes zu vermerken:

a) Bei Verwendung von UNBUFFERED ist die FROM-Option in das REWRITE-Statement einzuschließen.

b) Bei Verwendung von BUFFERED können noch die beiden folgenden Anweisungen zusätzlich zu den unter I bis IV. erwähnten eingesetzt werden:

```
V.   READ FILE(dateiname) SET(zeiger) [KEYTO(sv)] ;
VI.  LOCATE variable FILE(dateiname) KEYFROM(ausdruck) [SET(zeiger)] ;
```

1) SEDZ ist die Abkürzung von Speichereinheit mit direkter Zugriffsmöglichkeit. Dahinter verbergen sich solche Geräte wie Magnettrommelspeicher-, Magnetplattenspeichergeräte usw.

Die READ-Anweisung kann sowohl auf Dateien bezogen werden, die mit dem Attribut INPUT erklärt sind, als auch auf Dateien mit dem Attribut UPDATE. Die Verwendung der LOCATE-Anweisung ist auf Dateien beschränkt, die das Attribut OUTPUT besitzen.

Wenn eine regional organisierte Datenmenge mittels der sequentiellen Zugriffsmethode erschlossen wird, so werden die Sätze in der Reihenfolge zur Verfügung gestellt, in der sie in der Datenmenge angeordnet sind. In die Vereinbarung über den Dateinamen muß das Attribut KEYED einbezogen werden, wenn eine der Optionen KEYTO oder KEYFROM niedergeschrieben wird. Die in der KEYTO-Option angegebene Schlüsselvariable sv muß zum Zeichenkettentyp gehören.

Nach diesen kurzgefaßten Exkursionen über die sequentielle Zugriffsmethode wollen wir uns jetzt der direkten Zugriffsmethode zuwenden, die ja bekanntlich die Angabe des Attributes DIRECT erfordert. Hinzu muß notwendigerweise als weiteres Attribut KEYED kommen. Folgende Anweisungen bewirken dann einen direkten Zugriff zu Datensätzen:

```
 I. READ FILE(dateiname) INTO(variable) KEY(ausdruck) ;
II. REWRITE FILE(dateiname) FROM(variable) KEY(ausdruck) ;
III. DELETE FILE(dateiname) KEY(ausdruck) ;
IV. WRITE FILE(dateiname) FROM(variable) KEYFROM(ausdruck) ;
```

Die Benutzung des Statements DELETE ist allein im Betriebssystem OS erlaubt; es verursacht die Entfernung des betreffenden Satzes aus der Datenmenge. Das Attribut INPUT bedeutet, daß nur das READ-Statement eingesetzt werden darf; bei OUTPUT ist die alleinige Verwendung von WRITE obligatorisch. Ist dagegen eine Datei mit dem Attribut UPDATE erklärt, so dürfen alle vier Anweisungen benutzt werden.

Im ENVIRONMENT-Attribut ist die Art der regionalen Organisationsform stets festzulegen, gleichgültig, welche Zugriffsmethode verwendet wird. Infolgedessen ist eine der drei Angaben

```
REGIONAL(1)
REGIONAL(2)
REGIONAL(3)
```

aufzunehmen. Im Betriebssystem DOS ist die Angabe von REGIONAL(2) nicht zulässig. Bei Dateien, für die REGIONAL(3) angegeben ist, ist im Betriebssystem DOS darauf zu achten, daß im ENVIRONMENT-Attribut auch die Angabe KEYLENGTH(n) erscheint; nur auf diese Weise kann die Länge der auf die SEDZ aufgezeichneten Schlüssel festgehalten werden.

Bezüglich der OPEN- und CLOSE-Anweisungen gibt es bei regional organisierten Datenmengen keinen Unterschied gegenüber anderen Datenmengen.

Wir wollen nunmehr unser Augenmerk den Besonderheiten der einzelnen Varianten der regionalen Organisationsform schenken.

a) REGIONAL(1)
In eine dieser Organisationsform unterliegende Datenmenge können nur ungeblockte Sätze fester Länge (Format F) aufgenommen werden. Auf die Spuren der SEDZ werden keine Schlüssel aufgezeichnet. Durch die Datenelemente bzw. Ausdrücke in den Optionen KEY, KEYTO oder KEYFROM wird die relative Satznummer festgelegt. Diese muß achtstellig in Form einer aus Ziffern bestehenden Zeichenkette vorliegen und beginnt mit ØØØØØØØØ, ØØØØØØØ1 usw. In der KEYTO-Option muß bezug auf eine Zeichenkettenvariable genommen werden; bei den Optionen KEY und KEYFROM kann auch ein Ausdruck erscheinen. Dieser

wird in eine Zeichenkette umgewandelt. Die relative Satznummer darf keine eingebetteten Leerzeichen enthalten.

Nur die rechten acht Zeichen der in den Optionen auftretenden bzw. aus ihren Ausdrücken generierten Zeichenketten bilden die relative Satznummer. Wenn eine KEYTO-Option eine Zeichenkettenvariable mit einer größeren Länge als 8 spezifiziert, so wird die relative Satznummer rechtsbündig dieser Zeichenkettenvariablen zugewiesen, die im übrigen nach links mit Leerzeichen aufgefüllt wird. In den Ausdrücken der KEY- bzw. KEYFROM-Optionen können die führenden Nullen durch Leerzeichen ersetzt werden. Während der Erstellung einer Datei werden unbenutzte Regionen mit Scheinsätzen gefüllt, die die sedezimale Verschlüsselung FF im ersten Byte aufweisen. Beim Lesen einer Datei werden diese Scheinsätze genau so bereitgestellt wie die regulären Sätze. Es ist deshalb von eminenter Wichtigkeit, daß vor der Verarbeitung der Inhalt des ersten Bytes untersucht wird. - Im Betriebssystem OS bewirkt die DELETE-Anweisung, daß das erste Byte des Satzes auf FF (sedezimal) gesetzt wird, wodurch er als Scheinsatz markiert ist.

b) REGIONAL(2)
Eine nach dieser Variante regional organisierte Datenmenge enthält ungeblockte Sätze fester Länge (Satzformat F), die zusammen mit einem Schlüssel auf die Spuren der SEDZ aufgezeichnet werden. Die Länge des aufgezeichneten Schlüssels kann im DD-Statement der Jobsteuersprache unter KEYLEN aufgeführt werden. Die Optionen KEYTO, KEY bzw. KEYFROM beziehen sich in ihren rechten acht Stellen auf die relative Satznummer; die linksbündigen Bytes hingegen enthalten den aufgezeichneten bzw. aufzuzeichnenden Schlüssel. Ist die Länge der Zeichenkette größer als die Summe der Längen von relativer Satznummer und Aufzeichnungsschlüssel, so werden die mittleren Bytes nicht benutzt; ist sie dagegen kürzer, so werden einige mittlere Bytes sowohl dem Aufzeichnungsschlüssel als auch der relativen Satznummer zugeschlagen. Scheinsätze werden durch FF (sed) im ersten Byte des Aufzeichnungsschlüssels angezeigt. Die relative Satznummer dient allein zum Aufsuchen der durch sie angesprochenen Spur. Nachdem diese Spur gefunden ist, wird innerhalb der Spur der zu suchende Satz mit Hilfe des Aufzeichnungsschlüssels ermittelt. Durch den LIMCT-Parameter im DD-Statement der Jobsteuersprache kann das Suchen auf eine bestimmte Spurenanzahl erweitert werden.

c) REGIONAL(3)
In eine nach dieser Variante regional organisierte Datenmenge können sowohl Sätze des F- als auch des U-Formats hineingestellt werden [1]; im Betriebssystem OS sind darüberhinaus auch Sätze variabler Länge (V- bzw. S-Format) erlaubt. Geblockte Sätze können jedoch unter keinen Umständen benutzt werden. Die nach dieser Variante organisierten Datenmengen verhalten sich, von einer Ausnahme abgesehen, wie die nach der Variante 2 regional organisierten Datenmengen: Die rechten acht Bytes der bei den Optionen KEY, KEYTO bzw. KEYFROM stehenden bzw. generierten Zeichenketten werden nicht als relative Satznummer sondern als relative Spurnummer interpretiert. Scheinsätze werden nur erzeugt, wenn das DELETE-Statement auf existierende Sätze einwirkt. Damit aber ist klar gesagt, daß es im Betriebssystem DOS keine Scheinsätze geben kann, da unter diesem System keine Programme laufen können, die DELETE-Statements enthalten.

1) F-Format: Ungeblockte Sätze fester Länge.
U-Format: Sätze undefinierter Länge.

13.4. Datenmengen mit indexsequentieller Organisationsform

Sollen Datenmengen indexsequentiell organisiert werden, so ist es unumgänglich notwendig, für sie SEDZ bereitzustellen.

Bei der sequentiellen Zugriffsmethode (Dateiattribut SEQUENTIAL) können folgende Statements eingesetzt werden:

```
I.   READ FILE(dateiname) INTO(variable) [KEYTO(zkv)|KEY(ausdruck)] ;
II.  READ FILE(dateiname) SET(zeiger) [KEYTO(zkv)|KEY(ausdruck)] ;
III. READ FILE(dateiname) [IGNORE(ausdruck)] ;
IV.  REWRITE FILE(dateiname) [FROM(variable)] ;
V.   DELETE FILE(dateiname) [KEY(ausdruck)] ;
VI.  WRITE FILE(dateiname) FROM(variable) KEYFROM(ausdruck) ;
VII. LOCATE variable FILE(dateiname) KEYFROM(ausdruck) ;
```

Vom Dateiverwendungsattribut hängt ab, welche Statements verwendet werden dürfen:

a) INPUT ⟶ READ
b) OUTPUT ⟶ WRITE,LOCATE
c) UPDATE ⟶ READ,REWRITE,DELETE

Im Betriebssystem DOS ist die Benutzung des DELETE-Statements nicht möglich.

Bei der direkten Zugriffsmethode (Dateiattribut DIRECT) können folgende Statements eingesetzt werden:

```
I.   READ FILE(dateiname) INTO(variable) KEY(ausdruck) ;
II.  REWRITE FILE(dateiname) FROM(variable) KEY(ausdruck) ;
III. WRITE FILE(dateiname) FROM(variable) KEYFROM(ausdruck) ;
IV.  DELETE FILE(dateiname) KEY(ausdruck) ;
```

Dateien, die das Attribut INPUT besitzen, dürfen nur mit dem READ-Statement angesprochen werden. Auf Dateien mit dem Attribut UPDATE dürfen hingegen alle Anweisungen bezogen werden. Das Attribut OUTPUT in Verbindung mit dem Attribut DIRECT darf jedoch bei Dateien mit der indexsequentiellen Organisationsform nicht erscheinen. Wie auch bei der sequentiellen Zugriffsmethode ist im Betriebssystem DOS die Benutzung der DELETE-Anweisung nicht möglich.

Die Dateierklärung muß das Attribut KEYED einschließen, wenn in einem Statement eine der Optionen KEYTO, KEY bzw. KEYFROM benutzt wird. Im ENVIRONMENT-Attribut muß außerdem durch die Angabe INDEXED auf die indexsequentielle Organisationsform verwiesen werden.

Wenn der Schlüssel Teil des Satzes ist, so wird durch die Angabe KEYLOC(n) im ENVIRONMENT-Attribut die Position des Schlüssels innerhalb des Satzes festgelegt; n gibt dabei das Byte (relativ zu 1) an, auf dem der Schlüssel beginnt. Nicht zu berücksichtigen sind hierbei die vier Steuerbytes bei Sätzen im V- bzw. VB-Format. Beim Betriebssystem DOS ist die Angabe von KEYLOC unbedingt erforderlich, im Betriebssystem OS hingegen kann stattdessen auch das DD-Statement der Jobsteuersprache zur Festlegung der Schlüsselposition herangezogen werden. Falls generische Schlüssel bei einer Datenmenge eine Rolle spielen sollen, so muß im ENVIRONMENT-Attribut die Angabe GENKEY gemacht werden.

Nun mögen noch einige Notizen zu anderen Angaben des ENVIRONMENT-Attributes folgen. Wird zu den Sätzen einer Datei direkt zugegriffen, so kann durch die Angabe INDEXAREA ein Hauptspeichergebiet zur Aufnahme des Hauptinhaltsverzeichnisses oder eines Teiles

desselben angefordert werden. Dadurch, daß der Hauptindex bzw. ein Teil desselben im Hauptspeicher verbleibt, kann die Verarbeitung wesentlich beschleunigt werden. Durch NOWRITE wird festgelegt, daß keine WRITE-Anweisungen benutzt werden, wenn eine Datei mit den Attributen DIRECT und INPUT bzw. UPDATE vorliegt. Die Angabe ADDBUFF schließlich bewirkt die Bereitstellung von zusätzlichem Hauptspeicherplatz zwecks Neuanordnung der Sätze, wenn eine Datei durch neu hinzugekommene Sätze erweitert werden soll. Die Benutzung von ADDBUFF im Betriebssystem OS setzt voraus, daß die Datei mit den Attributen DIRECT sowie INPUT bzw. UPDATE deklariert ist und aus Sätzen fester Länge (F-Format) besteht; bei Sätzen variabler Länge (V-Format) wird dieses zusätzliche Hauptspeichergebiet automatisch zugeordnet.

Im Betriebssystem DOS ist die Angabe von INDEXMULTIPLE im ENVIRONMENT-Attribut immer dann erforderlich, wenn das Zylinderinhaltsverzeichnis auf einem anderen EA-Gerät existiert als auf dem in der MEDIUM-Angabe spezifizierten. Das gleiche gilt, wenn ein Hauptinhaltsverzeichnis benutzt wird. Auch erfordert das Betriebssystem DOS die Angabe OFLTRACKS bei einer mit dem Attribut OUTPUT versehenen Datei, um die Anzahl der Überlaufspuren festzulegen, die in jedem Zylinder zu reservieren sind. Im Betriebssystem DOS sind allein nur die Satzformate F und FB zulässig; das Betriebssystem OS erlaubt zusätzlich noch die Satzformate V und VB.

Die Anweisung DELETE ist nur im Betriebssystem OS erlaubt, wir sagten das schon. Ergänzend müssen wir dazu bemerken, daß außerdem zusätzliche Voraussetzungen einzuhalten sind. Die Datenmenge muß entweder unter Angabe von OPTCD=L im DD-Statement der Jobsteuersprache erstellt sein oder sie enthält nur ungeblockte Sätze ohne eingebettete Schlüssel. Tatsächlich findet kein physisches Entfernen von Sätzen statt; die durch die DELETE-Anweisung angesprochenen Sätze werden vielmehr nur durch FF (sed) auf dem ersten Byte gekennzeichnet.

Die Anweisungen OPEN und CLOSE werden auch bei Dateien, die der indexsequentiellen Organisationsform unterliegen, in der üblichen Weise benutzt.

Wenn die sequentielle Zugriffsmethode nur auf einen Teil einer Datei bezogen werden soll, so ist die Option KEY zu benutzen. Der in ihr stehende Ausdruck wird ausgewertet und in eine Zeichenkette umgewandelt. Diese wird dann als Schlüssel des Satzes angesehen, zu dem zuerst zugegriffen werden soll. Nachfolgende READ-Anweisungen ohne die KEY-Option greifen dann zu den sich anschließenden Sätzen zu. Ergibt sich aus der KEY-Option ein Schlüssel, der eine kürzere Länge besitzt als die aufgezeichneten Schlüssel aufweisen, so wird bei Fehlen der Angabe GENKEY im ENVIRONMENT-Attribut der ermittelte Schlüssel nach rechts mit Leerzeichen aufgefüllt. Bei Angabe von GENKEY wird der ermittelte Schlüssel als generischer Schlüssel angesehen. Doch zurück zum Fall, daß GENKEY fehlt. Bei einer solchen Schlüsselergänzung kann es natürlich, ja wird es in der Mehrzahl der Fälle sogar, vorkommen, daß ein Satz mit dem ermittelten Schlüssel nicht gefunden wird. Es kommt dann zum Auftreten der Bedingung KEY; nachfolgende Leseoperationen werden davon jedoch nicht betroffen. — Die Option KEYTO erfordert die Aufführung einer Zeichenkettenvariablen, hier durch zkv angedeutet. Dieser wird im Zuge der Leseoperation der Schlüssel des gelesenen Satzes zugewiesen.

Bei der LOCATE-Anweisung wird der Wert in der KEYFROM-Option erst benutzt, wenn der Satz tatsächlich durch ein WRITE-, CLOSE- oder das nachfolgende LOCATE-Statement geschrieben wird.

13.5. Dateien mit den von VSAM unterstützten Dateiorganisationsformen

Dateien in Zugangsfolge werden in der gleichen Weise vereinbart wie Dateien, die der konsekutiven Organisationsform unterliegen. Es ist allerdings unerläßlich, in die Dateierklärung das Attribut ENVIRONMENT mit der Angabe VSAM aufzunehmen. Bezüglich der EA-Anweisungen ist zu sagen, daß genau die gleichen benutzt werden können wie die, die wir für die sequentielle Zugriffsmethode bei Dateien mit konsekutiver Organisationsform kennengelernt haben.

Dateien in Schlüsselfolge erfordern im ENVIRONMENT-Attribut die Angabe VSAM. Hinzu kommt INDEXED; weitere allgemeine Angaben wie GENKEY und COBOL können ebenfalls vorhanden sein.

Für die sequentielle Zugriffsmethode (Attribut SEQUENTIAL) existieren dann die folgenden EA-Anweisungen:

```
I.   READ FILE(dateiname) INTO(variable) [KEYTO(zkv)|KEY(ausdruck)] ;
II.  READ FILE(dateiname) SET(zeiger) [KEYTO(zkv)|KEY(ausdruck)] ;
III. READ FILE(dateiname) [IGNORE(ausdruck)] ;
IV.  REWRITE FILE(dateiname) [FROM(variable)] ;
V.   DELETE FILE(dateiname) [KEY(ausdruck)] ;
VI.  WRITE FILE(dateiname) FROM(variable) KEYFROM(ausdruck) ;
VII. LOCATE variable FILE(dateiname) KEY(ausdruck) [SET(zeiger)] ;
```

Das dem Dateinamen zugeordnete Dateiverwendungsattribut bestimmt, welche Anweisungen benutzt werden dürfen. Es gilt:

a) INPUT
   Alle Arten der READ-Anweisungen, die SET-Option jedoch nicht bei Unterlassung der Pufferung (Attribut UNBUFFERED).

b) OUTPUT
   WRITE-Anweisung und bei Pufferung (Attribut BUFFERED) auch die LOCATE-Anweisung.

c) UPDATE
   Alle Anweisungen mit Ausnahme der WRITE- und LOCATE-Anweisung, die READ-Anweisung mit der SET-Option ist jedoch bei ungepufferten Dateien (Attribut UNBUFFERED) nicht erlaubt.

Generell ist für ungepufferte Dateien (Attribut UNBUFFERED) zu sagen, daß die READ-Anweisung mit der SET-Option und die LOCATE-Anweisung nicht codiert werden dürfen. Außerdem ist erforderlich, die REWRITE-Anweisung mit der FROM-Option niederzuschreiben.

Wenden wir uns den Statementformen zu, die wir zur Realisierung der direkten Zugriffsmethode (Attribut DIRECT) benutzen können. Sie lauten:

```
I.   READ FILE(dateiname) INTO(variable) KEY(ausdruck) ;
II.  REWRITE FILE(dateiname) FROM(variable) KEY(ausdruck) ;
III. DELETE FILE(dateiname) KEY(ausdruck) ;
IV.  WRITE FILE(dateiname) FROM(variable) KEYFROM(ausdruck) ;
```

Das Attribut INPUT läßt nur die Verwendung von READ-Statements zu, das Attribut UPDATE alle Statements. Die Benutzung von OUTPUT als Dateiverwendungsattribut ist nicht möglich.

Allgemein ist zu sagen, daß VSAM an Stelle der indexsequentiellen Organisationsform benutzt werden sollte, da die Verarbeitungsgeschwindigkeit gegenüber dieser anwächst. Man halte sich jedoch vor Augen, daß nicht alle Installationen VSAM unterstützen, so daß bei ihnen auf die Benutzung von VSAM verzichtet werden muß.

13.6. Ein Beispiel für die Direktverarbeitung

Die Abb. 13.1. zeigt uns ein PL/1-Programm, in dem die direkte Zugriffsmethode an zwei Dateien demonstriert wird. Eine Datei ist dabei indexsequentiell, die andere regional (Variante 1) organisiert.

```
BP13Ø1:  PROCEDURE OPTIONS(MAIN) ;
           DECLARE  TEILE FILE RECORD UPDATE DIRECT KEYED
                       ENVIRONMENT (INDEXED F BLKSIZE(24Ø) RECSIZE(48)
                                    KEYLENGTH(7) KEYLOC(34)
                                    MEDIUM(SYSØØ8,2314)),
                    UFA FILE RECORD UPDATE DIRECT KEYED
                       ENVIRONMENT (REGIONAL(1) F BLKSIZE(4Ø)
                                    MEDIUM(SYSØØ6,2314)),
                    EIN FILE RECORD INPUT SEQUENTIAL BUFFERED
                       ENVIRONMENT (F BLKSIZE(8Ø) BUFFERS(2)
                                    MEDIUM(SYSIPT,254Ø)),
                    LKAUS FILE RECORD OUTPUT SEQUENTIAL BUFFERED
                       ENVIRONMENT (F BLKSIZE(8Ø) BUFFERS(2)
                                    MEDIUM(SYSPCH,254Ø)) ;
           DECLARE  1 TSATZ,    /* SATZ DER DATEI 'TEILE' */
                      2 BEZ CHAR(28),    /* TEILEBEZEICHNUNG */
                      2 MEN FIXED(9),    /* TEILEMENGE */
                      2 TNR PIC'(7)9',   /* TEILENUMMER */
                      2 VWW PIC'(8)9',   /* VERWEIS */
                    1 USATZ,    /* SATZ DER DATEI 'UFA' */
                      2 AUF CHAR(14),
                      2 DATUM CHAR(8),
                      2 KDNR PIC'(5)9',
                      2 MENGE FIXED(9),
                      2 VW PIC'(8)9',
                    1 LSATZ,    /* SATZ DER DATEIEN 'EIN' UND 'LKAUS' */
                      2 MENGE CHAR(1Ø),
                      2 LZ1 CHAR(1),
                      2 TEILENR CHAR(7),
                      2 LZ2 CHAR(1),
```

```
                    2 BEZE CHAR(28),
                    2 LZ3 CHAR(1),
                    2 KDNR CHAR(5),
                    2 LZ4 CHAR(1),
                    2 AUF CHAR(14),
                    2 LZ5 CHAR(1),
                    2 DATUM CHAR(8),
                    2 LZ6 CHAR(1),
                    2 KZ CHAR(1),
                    2 TYP CHAR(1),
                  1 NS,
                    2 MENGE FIXED(9),
                    2 TNR PIC'(7)9',
                    2 KDNR PIC'(5)9',
                  (VØ,VLSATZ) PIC'(8)9',
                  ZTW FIXED(9),
                  MG PIC'(9)Z9',
                  Z PIC'(8)9' INITIAL(Ø)) ;
START: OPEN FILE(TEILE), FILE(UFA), FILE(EIN), FILE(LKAUS) ;
       ON ENDFILE(EIN)  GOTO N4 ;
       ON KEY(TEILE)  GOTO M1 ;
       ON CONVERSION  GOTO M1 ;
       READ FILE(UFA) INTO(USATZ) KEY(Z) ;
       VØ = USATZ.VW ;
       GOTO M2 ;
ENDE:  UFA.VW = VØ ;
       REWRITE FILE(UFA) FROM(USATZ) KEY(Z) ;
       CLOSE FILE(TEILE), FILE(UFA), FILE(EIN), FILE(LKAUS) ;
       RETURN ;
  M1:  WRITE FILE(LKAUS) FROM(LSATZ) ;
  M2:  READ FILE(EIN) INTO(LSATZ) ;
       LSATZ.KZ = '1' ;
       GET STRING(LSATZ.MENGE) EDIT(NS.MENGE)
                                   (F(1Ø)) ;
       IF  NS.MENGE<Ø  THEN GOTO M1 ;
       LSATZ.KZ = '2' ;
       GET STRING(LSATZ.TEILENR) EDIT(ZTW) (F(7)) ;
       IF  ZTW<Ø  THEN GOTO M1 ;
       NS.TNR = ZTW ;
```

```
    NS.KDNR = Ø ;
    IF  LSATZ.KDNR=(5)' '  THEN  GOTO M4 ;
    LSATZ.KZ = '3' ;
    GET STRING(LSATZ.KDNR) EDIT(ZTW) (F(5)) ;
    IF  ZTW<Ø  THEN GOTO M1 ;
    NS.KDNR = ZTW ;
M4: LSATZ.KZ = '4' ;
    IF  LSATZ.TYP<'3' | LSATZ.TYP>'5'  THEN GOTO M1 ;
    LSATZ.KZ = '5' ;
    IF  LSATZ.TYP='3' & LSATZ.KDNR=(5)' '  THEN GOTO M1 ;
    LSATZ.KZ = '6' ;
    READ FILE(TEILE) INTO(TSATZ) KEY(NS.TNR) ;
    LSATZ.BEZE = TSATZ.BEZ ;
    LSATZ.KZ = ' ' ;
    LZ1 = ' ' ;
    LZ2 = ' ' ;
    LZ3 = ' ' ;
    LZ4 = ' ' ;
    LZ5 = ' ' ;
    LZ6 = ' ' ;
    IF  LSATZ.TYP¬='3'  THEN GOTO N1 ;
L1: VLSATZ = TSATZ.VWN ;
    IF  TSATZ.VWN¬=Ø  THEN GOTO L5 ;
    IF  TSATZ.MEN<NS.MENGE  THEN GOTO L6 ;
    LSATZ.TYP = '7' ;
    TSATZ.MEN = TSATZ.MEN - NS.MENGE ;
L2: WRITE FILE(LKAUS) FROM(LSATZ) ;
L3: REWRITE FILE(TEILE) FROM(TSATZ) KEY(NS.TNR) ;
    GOTO M2 ;
L4: VLSATZ = USATZ.VW ;
L5: READ FILE(UFA) INTO(USATZ) KEY(VLSATZ) ;
    IF  USATZ.VW¬=Ø  THEN GOTO L4 ;
    USATZ.VW = VØ ;
    REWRITE FILE(UFA) FROM(USATZ) KEY(VLSATZ) ;
    GOTO L7 ;
L6: MG = NS.MENGE - TSATZ.MEN ;
    LSATZ.MENGE = MG ;
    TSATZ.VWN = VØ ;
```

```
L7: VLSATZ = VØ ;
    READ FILE(UFA) INTO(USATZ) KEY(VLSATZ) ;
    VØ = USATZ.VW ;
    USATZ.VW = Ø ;
    USATZ.AUF = LSATZ.AUF ;
    USATZ.DATUM = LSATZ.DATUM ;
    USATZ.KDNR = NS.KDNR ;
    USATZ.MENGE = NS.MENGE ;
    LSATZ.TYP = '6' ;
    REWRITE FILE(UFA) FROM(USATZ) KEY(VLSATZ) ;
    GOTO L2 ;
N1: IF  LSATZ.TYP='4'  THEN  TSATZ.MEN = TSATZ.MEN + NS.MENGE
                       ELSE  TSATZ.MEN = NS.MENGE ;
N3: IF  USATZ.VW=Ø  THEN  GOTO L3 ;
    VLSATZ = TSATZ.VWW ;
    READ FILE(UFA) INTO(USATZ) KEY(VLSATZ) ;
    IF  TSATZ.MEN<USATZ.MENGE  THEN GOTO L3 ;
    TSATZ.MEN = TSATZ.MEN - USATZ.MENGE ;
    MG = USATZ.MENGE ;
    LSATZ.MENGE = MG ;
    LSATZ.KDNR = USATZ.KDNR ;
    LSATZ.AUF = USATZ.AUF ;
    LSATZ.DATUM = USATZ.DATUM ;
    TSATZ.VWW = USATZ.VW ;
    USATZ.VW = VØ ;
    VØ = VLSATZ ;
    REWRITE FILE(UFA) FROM(USATZ) KEY(VLSATZ) ;
    WRITE FILE(LKAUS) FROM(LSATZ) ;
    GOTO N3 ;
N4: DISPLAY ('PROGRAMMENDE') ;
    GOTO ENDE ;
  END BP13Ø1 ;
```

Abb. 13.1. Beispielprogramm für die direkte Zugriffsmethode (Betriebssystem DOS)

Die Teiledatei mit dem Namen TEILE enthält Informationen über den Bestand an Teilen; außerdem werden die Teile beschrieben. Hinzu kommt ferner ein Verweis auf einen Satz in der Datei UFA. Die Datei UFA enthält Informationen über unerfüllte Aufträge, d.h. über Bestellungen, die wegen eines unzureichenden Lagerbestandes nicht ausgeliefert werden konnten. Die Datei TEILE ist indexsequentiell organisiert, als Schlüssel wird die Teilenummer TNR benutzt. Die Datei UFA ist hingegen regional organisiert.

Bei der in einem gesonderten Programm vorgenommenen Erstellung der Datei UFA wird jeder Satz mit einem Verweis VW geschrieben; VW zeigt dabei auf den nächsten verfügbaren Satz in der Datei. So hat z.B. der Satz mit der Nummer 00000034 den Verweis 00000035 usw. Wenn nun ein Auftrag zurückgestellt werden muß, wird das Verweisfeld VW im Teilesatz der Datei TEILE von 00000000 geändert, d.h. ihm wird ein Wert zugewiesen, der gleich der Satznummer des ersten verfügbaren Satzes in der Datei UFA ist. Dieser Satz wird dann mit den Daten des zurückgestellten Auftrages gefüllt und außerdem sein Verweisfeld auf 00000000 gesetzt. Kommt es zu einem zweiten nicht ausführbaren Auftrag, so wird dem Verweisfeld VW des Satzes von UFA, der die Daten des ersten nicht ausgeführten Auftrages enthält, die Satznummer des nächsten in UFA verfügbaren Satzes zugewiesen. In den Satz selbst werden die Daten des zweiten nicht ausführbaren Auftrages hineingestellt und sein Verweisfeld auf 00000000 gesetzt. Der Satz 00000000, d.h. der erste Satz in der Datei UFA verweist stets auf den ersten verfügbaren Satz in UFA. Dieser Wert wird zu Programmbeginn in den Hauptspeicher eingelesen und zu Programmschluß (ENDE) zurückgeschrieben.

Aufträge, wie auch Wareneingänge und auf Zählungen basierende Bestandskorrekturen werden von Lochkarten gelesen, die über die Eingabedatei EIN einlaufen. Versandbestellungen, Aufträge, die mit dem vorhandenen Lagerbestand nicht realisiert werden können und der Inhalt fehlerhafter Lochkarten werden auf Lochkarten ausgegeben, und zwar über die Datei LKAUS. Da die Möglichkeit, daß fehlerhaft abgelochte Lochkarten gelesen werden, durchaus gegeben ist, wird der Inhalt der Lochkarten sorgfältig geprüft: Jeder festgestellte Fehler bewirkt eine Verzweigung zum Statement, das mit der Marke M1 markiert ist. Von hier aus wird das Stanzen einer Fehlerkarte veranlaßt; diese kann an Hand eines vom Leerzeichen verschiedenen Zeichens in Spalte 79 (Variable KZ) identifiziert werden. Die Verarbeitung fehlerhafter Lochkarten wird selbstverständlich verhindert. Das Auftreten irgendeines Umwandlungsfehlers (CONVERSION-Bedingung) bewirkt gleichfalls eine Verzweigung nach M1. Die Angabe einer falschen Teilenummer TNR hat das Entstehen der KEY-Bedingung in dem Augenblick zur Folge, in dem ein Teilesatz aus der Datei TEILE gelesen werden soll. Auch hier wird eine Verzweigung nach M1 eingeleitet.

Man beachte, daß ein großer Teil des Programmes, nämlich der zwischen M1 und L1 liegende, für Plausibilitätsprüfungen der Eingabewerte geschaffen ist, ein typisches Zeichen bei vielen kommerziell orientierten Programmen. Die Statementfolge ab L1 verarbeitet die Aufträge, wobei in der Konsequenz entweder Lieferscheine ausgestellt oder Sätze mit Auftragsrückstellungen in die Datei UFA geschrieben werden.

Der Programmabschnitt ab N1 dient dazu, Wareneingänge oder Lagerbestandskorrekturen zu verarbeiten. Nach Errechnung des neuen Lagerbestandes wird der Versuch unternommen, den ersten zurückgestellten Auftrag zu befriedigen, falls ein solcher vorhanden ist. Gelingt das, so wird der entsprechende Satz von UFA wieder freigegeben und der Versuch unternommen, weitere zurückgestellte Aufträge auszuführen.

Wenn Beziehungen zwischen Dateien bestehen, die denen unseres Beispielprogrammes entsprechen, so spricht man von angehängten Dateien. Hier ist also die Datei UFA eine an die Datei TEILE angehängte Datei. Die Sätze der Datei EIN wiederum sind die Ursache für das Lesen von Sätzen in der Datei TEILE. Eine Datei, die Veranlassung gibt, zu einer anderen Datei zuzugreifen, heißt allgemein Auftraggeberdatei.

Durch dieses eine in Abb. 13.1. vorgestellte Programm kann natürlich das gesamte Verfahren zur Auftragsabwicklung nicht voll befriedigt werden. Es gehören noch andere Programme dazu, die veraltete Teilesätze entfernen, die neue Teilesätze hinzufügen u.a.m.; beispielsweise wäre auch ein Programm sinnvoll, das eine Aufstellung über alle ausstehenden nicht erfüllten Bestellungen druckt.

Aufgabengruppe 33

1) Besonders wichtig sind Dateien, zu denen direkt zugegriffen werden kann, wenn es sich um Arbeiten mit umfangreichen Matrizen handelt. Es ist ein Programm zu schreiben, das Lochkarten liest, auf denen die Werte von Matrixelementen in folgender Form abgelocht sind:

Spalten 1 und 2:	Zeilenindex i des Matrixelementes
Spalten 3 und 4:	Spaltenindex j des Matrixelementes
Spalten 5 bis 12:	Wert des Matrixelementes a_{ij}

Für i und j gelten die Ungleichungen $i<100$ bzw. $j<100$. Vor dem Lesen der ersten Lochkarte mit dem Wert eines Matrixelementes soll eine Vorlaufkarte eingelesen werden, die wie folgt abgelocht ist:

Spalten 1 und 2:	Zeilenzahl m der Matrix
Spalten 3 und 4:	Spaltenzahl n der Matrix

Auf diese Vorlaufkarte folgen dann die $m \cdot n$ Lochkarten mit den Werten der Matrixelemente in einer beliebigen Reihenfolge. Nach dem Lesen ist jeder Wert sofort in eine Datei zu schreiben, wobei die direkte Zugriffsmethode vorzusehen ist. Die Werte der Matrixelemente sollen in der Reihenfolge

$$a_{11}, a_{12}, \cdots, a_{1n}, a_{21}, \cdots, a_{mn}$$

in der Datei angeordnet werden. Nach dem Einlesen der letzten Datenkarte sind die Matrixelemente in der Speicherungsreihenfolge zu lesen und ihre Werte zu drucken.

2) Schreibe ein Programm, das die im Programm von Aufg. 1) erstellte Datei liest und die gemäß nachstehender Beziehungen ausgewählten Werte in eine Matrix $\mathcal{B}(k,l)$ stellt, deren Elemente wie folgt gebildet werden:

$$k = m-7, m-6, \cdots, m-1, m$$
$$l = 4, 5, 6, \cdots, n$$

Nach Voraussetzung ist für diesen Fall $n<20$.

3) Eine Hochschule plant die Speicherung ihres Vorlesungsverzeichnisses auf Magnetbändern. Die zur Erstellung dieser Datei erforderlichen Informationen sind auf Lochkarten wie folgt abgelocht:

Spalten 1 und 2:	Fakultätsnummer
Spalten 3 und 4:	Abteilungsnummer
Spalten 5 bis 7:	Vorlesungsnummer
Spalten 8 bis 27:	Titel der Vorlesung
Spalten 28 und 29:	Vorlesungszeit (Beginn der Vorlesung)
Spalten 30 bis 35:	Vorlesungstage (mit folgenden Abkürzungen: M, D, I, D, F, S)
Spalten 36 bis 38:	Maximale Teilnehmerzahl

Es ist ein Programm zu schreiben, das diese Karten liest und ihren Inhalt als Sätze auf einem Magnetband speichert. Die Sätze sollen wie folgt aufgebaut sein:

Stellen 1 und 2:	Fakultätsnummer
Stellen 3 und 4:	Abteilungsnummer
Stellen 5 bis 7:	Vorlesungsnummer
Stellen 8 bis 27:	Titel der Vorlesung
Stellen 28 und 29:	Vorlesungszeit
Stelle 30 :	Vorlesungstage (in einer Bitkette verschlüsselt)

Stellen 31 und 32: Maximal mögliche Teilnehmerzahl (dezimal in Festpunktdarstellung)
Stellen 33 und 34: Gegenwärtiger Einschreibungsstand (dezimal in Festpunktdarstellung mit dem Anfangswert Null)

Jeweils fünf Sätze sollen zu einem Block zusammengefaßt werden.

4) Es soll ein Programm geschrieben werden, mit dem die Vorlesungsbelegungen von Studenten überprüft werden können.

Zur Eingabe gelangt das Magnetband, das gemäß Aufgabe 3) erstellt worden ist. Die Angaben der Studenten sind auf Karten abgelocht, die wie folgt eingeteilt sind:

Spalten 1 bis 6: Immatrikulationsnummer des Studenten
Spalten 7 bis 46: Name
Spalten 47 bis 53:
Spalten 54 bis 60:
Spalten 61 bis 67:
Spalten 68 bis 74:
} Jeweils Fakultätsschlüssel, Abteilungsschlüssel und Vorlesungsschlüssel einer Vorlesung

Da ein Student maximal zwölf Vorlesungen belegen kann, können für jeden Studenten bis zu drei Karten vorliegen.

Im Programm sollen nun die Magnetbandsätze auf die belegten Vorlesungen durchsucht werden. Die entsprechenden Sätze sind zu lesen. Nach dem Lesen der Sätze aller angegebenen Vorlesungen ist zu überprüfen, ob sich Widersprüche beim Beginn der Vorlesungen ergeben, d.h. daß zwei Vorlesungen zur gleichen Zeit anfangen. Werden derartige Widersprüche gefunden, so soll der Vorlesungsplan des betreffenden Studenten in einer übersichtlichen Form gedruckt werden. Der Plan soll folgende Informationen enthalten: Immatrikulationsnummer, Name und für jede Vorlesung Fakultätsnummer, Abteilungsnummer, Vorlesungsnummer, Vorlesungstitel und Vorlesungszeit. Die Vorlesungszeit ist durch die Uhrzeit (08-20) des Vorlesungsbeginns festgehalten.

5) Das Programm von Aufgabe 3) ist wie folgt abzuändern: Die Ausgabe soll in eine Datei erfolgen, die der Organisationsform VSAM unterliegt. Der Schlüssel ist aus

— Fakultätsnummer
— Abteilungsnummer
— Vorlesungsnummer

zusammenzustellen.

6) Schreibe ein Programm, das die bei der Aufg. 4) beschriebenen Lochkarten liest und überprüfen soll, inwieweit überfüllte Vorlesungen vorliegen. Wenn ein Student sich nicht für eine überfüllte Vorlesung eingeschrieben hat, so ist eine 1 zum gegenwärtigen Einschreibungsstand der jeweiligen Vorlesung zu addieren und der Vorlesungsplan des Studenten zu drucken. Bei Feststellung einer überfüllten Vorlesung soll die Einschreibung des Studenten vorerst unterbunden werden. Zur besseren Übersicht ist ebenfalls sein Vorlesungsplan zu drucken; bei überfüllten Vorlesungen soll er jedoch bei der betreffenden Vorlesung mit dem zusätzlichen Vermerk

UEBERFUELLT

versehen werden. Es ist die in Aufg. 5) erstellte Datei anzusprechen.

13.7. BEGIN-Blöcke (Einlaufblöcke)

Ein Einlaufblock oder BEGIN-Block besitzt die folgende Gestalt:

```
[marke:]  BEGIN ;
          statement(s)
          END  marke  ;
```

Der BEGIN-Block wird also genau so wie eine einfache DO-Gruppe geschrieben und kann überall dort in einem Programm erscheinen, wo auch ein einfaches Statement auftreten darf. Oft wird er in einem ON-Statement oder in einem IF-Statement codiert werden, aber er kann natürlich auch im Hauptteil eines Programmes verwendet werden.

Vor dem BEGIN-Statement ist nicht unbedingt eine Marke erforderlich, es sei denn, daß dieser Block angesprungen wird. Ebenso braucht keine Marke im END-Statement erscheinen, es sei denn, daß dieses Statement mehr als einen Block oder eine Gruppe abschließen soll.

Der Unterschied zwischen einer DO-Gruppe und einem BEGIN-Block ist allein im Gültigkeitsbereich der Namen zu suchen. Wenn in einem DECLARE-Statement innerhalb einer DO-Gruppe ein Name erklärt wird, dann ist dieser Name im ganzen Programm bekannt und kann in ihm auch überall verwendet werden. Für einen in einem BEGIN-Block erklärten Namen gilt das nicht. Bezeichnungen, die in einem BEGIN-Block erklärt werden, sind nur innerhalb dieses Blockes bekannt; die Namen jedoch, die in der Prozedur außerhalb dieses Blockes erklärt sind, sind auch im Blockinnern gültig, es sei denn, daß der gleiche Name im Block selbst neu definiert wird.

Wir wollen ein Beispiel verfolgen (Abb. 13.2.).

```
ABC:  PROCEDURE ;
        DCL  X CHARACTER(5),
             Y DECIMAL FLOAT(6),
             Z DECIMAL FIXED(7,2) ;
        .
        .
        IF  Y>Ø  THEN GOTO EBL ;
        .
        .
EBL:    BEGIN ;
          DCL  U CHARACTER(6),
               V BINARY FIXED(31),
               W BIT(4),
               X DECIMAL FIXED(4) ;
          .
          .
        END EBL ;
        .
        .
      END ABC ;
```

Abb. 13.2. Beispiel für Gültigkeitsbereiche von Namen bei Einlaufblöcken

Innerhalb des BEGIN-Blockes sind die Bezeichnungen U, V, W und X definiert. Außerhalb dieses Blockes sind die Bezeichnungen X, Y und Z erklärt. Im Hauptteil des Programmes mit Ausnahme des BEGIN-Blockes sind also die Namen X (Name einer Zeichenkettenvariablen), Y und Z bekannt und dürfen deshalb dort verwendet werden.

Innerhalb des BEGIN-Blockes sind die Namen U, V und W genau so wie X(hier jedoch als arithmetische Variable in Festpunktdarstellung) bekannt und können deshalb verwendet werden; es sind aber auch die Namen Y und Z weiterhin bekannt und können deshalb auch eingesetzt werden. Nur die Zeichenkettenvariable X, die außerhalb dieses Blockes erklärt ist, ist im Innern dieses Blockes unbekannt und kann deshalb darin auch nicht angesprochen werden.

BEGIN-Blöcke können bei der Behandlung von ON-Bedingungen besonders nützlich sein. Solange wie in einem solchen Block nur die eigens darin definierten Variablen angesprochen werden, solange werden auch nicht die Variablen berührt, die außerhalb des Blockes im Programm eingeführt sind und dort auch verwendet werden.

BEGIN-Blöcke können genau so wie Gruppen ineinander verschachtelt sein. Die Regeln für den Gültigkeitsbereich eines jeden Namens werden grundsätzlich auf den Eingangspunkt eines jeden Blockes bezogen. Es wird keine Unterscheidung von der Stelle abhängig gemacht, an der ein Name bekannt gemacht wird.

Mitunter ist es sehr vorteilhaft, wenn eine in einem Block erklärte Variable auch außerhalb des Blockes bekannt ist. Fügt man der Erklärung einer solchen Variablen das Attribut EXTERNAL hinzu, so erreicht man damit, daß die Variable auch außerhalb des Blockes bekannt ist, in dem sie erklärt wurde. Hierzu geben wir zur Erläuterung noch einmal ein Beispiel:

```
DCL  U CHAR(6) EXTERNAL ;
```

Damit ist U als externer Name festgelegt und somit innerhalb und außerhalb des Blockes bekannt, in dem dieses DECLARE-Statement aufgeführt ist. Außerhalb des Blockes bedeutet, daß die Variable in jedem anderen Block bekannt ist, der den Block enthält, in dem die Variable erklärt ist.

Ein BEGIN-Block wird normal unter folgenden Konditionen verlassen:

a) Die Programmablaufsteuerung stößt auf das den Block beschließende END-Statement,

b) Ausführung einer RETURN-Anweisung,

c) Ausführung einer STOP-Anweisung.

Darüberhinaus kann ein Einlaufblock auch anormal durch eine GOTO-Anweisung verlassen werden, die zu einer Marke außerhalb des Blockes verzweigt. Das anormale Verlassen eines BEGIN-Blockes sollte weitgehend vermieden werden, da hierdurch im allgemeinen recht leistungsschwache Programme entstehen.

Ein BEGIN-Block wird in der Regel durch eine lineare Statementfolge betreten, d.h. nach Ausführung der unmittelbar vor dem BEGIN-Statement liegenden Anweisung geht die Programmablaufsteuerung zum BEGIN-Statement über. Ein BEGIN-Block kann daneben auch durch eine auf das BEGIN-Statement verzweigende Sprunganweisung angesteuert werden. Von diesen Grundsatzregeln weicht jeder als Oneinheit eingesetzte BEGIN-Block ab. Ein solcher Block wird nämlich nur dann angesprungen, wenn die in der ON-Anweisung genannte Bedingung aufgetreten ist. Eine Erweiterung der bisher bekannten Sprachregeln ist damit stillschweigend schon eingeführt worden. Die einfache Anweisung, die als Oneinheit codiert werden kann, ist stets als BEGIN-Block anzusehen.

13.8. Hauptspeicherklassen

Für eine Bezeichnung kann bekanntlich ihr Gültigkeitsbereich durch eines der beiden Attribute

INTERNAL

bzw.

EXTERNAL

festgelegt werden.

Eine mit dem Attribut INTERNAL versehene Bezeichnung repräsentiert eine Größe, die allein in dem Block [1] bekannt ist, in dem sie erklärt ist. Hinzu kommen als Gültigkeitsbereich diejenigen Blöcke, die in dem Block enthalten sind, in dem die Erklärung abgegeben wurde, und die selbst keine neue Erklärung der betreffenden Bezeichnung enthalten. Eine externe Bezeichnung stellt eine Größe dar, die nicht nur in dem Block bekannt ist, in dem sie erklärt ist, sondern auch in allen anderen Blöcken, selbst wenn sie nicht im ersten Block enthalten sind, in denen die gleiche Erklärung vorliegt.

Ist ein Block vollständig in einem anderen Block enthalten, so sagt man, daß der betreffende Block ein interner Block ist.

Ist eine Struktur mit dem Attribut EXTERNAL erklärt, so gelten die Namen der Strukturelemente und der Unterstrukturen als interne Namen, d.h. ihnen ist das Attribut INTERNAL zugeordnet. Nur der Name der Hauptstruktur ist also außerhalb des Blockes bekannt. Strukturelemente und Unterstrukturen können niemals mit dem Attribut EXTERNAL versehen werden.

Für Dateinamen und für die Namen der vom Programmierer benannten Bedingungen gilt standardmäßig das Attribut EXTERNAL, für alle anderen Namen das Attribut INTERNAL. Die Namen externer Prozeduren gelten ebenfalls als mit dem Attribut EXTERNAL ausgezeichnet.

Eine Größe kann zu einer der folgenden Hauptspeicherklassen gehören:

AUTOMATIC
STATIC
BASED

Diese drei Attribute bestimmen die Art und Weise, wie Variablen Hauptspeicherplatz zugeordnet wird, wenn ein Programm zur Ausführung gelangt. Variablen mit dem Attribut AUTOMATIC wird Speicherplatz zugeordnet, wenn ein Block betreten wird; nach Verlassen des Blockes wird der Speicherplatz wieder freigegeben. Für Variable mit dem Attribut STATIC wird der Speicherplatz bereits beim Laden des Programmes zugeordnet. Diese Zuordnung bleibt bis zum Programmschluß erhalten, also nicht nur bis zum Blockende. Für Variable mit dem Speicherklassenattribut BASED wird Speicherplatz nur in Folge der Ausführung von READ-Anweisungen mit der SET-Option bzw. von LOCATE-Anweisungen zugeordnet. Die Speicherplatzfreigabe erfolgt andererseits sofort beim Einleiten einer nachfolgenden EA-Operation, die auf die betreffende Datei bezug nimmt. Man beachte hierbei, daß die Zuordnung einer Zeigervariablen in der Erklärung einer Variablen nicht etwa schon die Zuordnung von Hauptspeicherplatz bedeutet. Diese erfolgt vielmehr erst im Moment der Ausführung der soeben erwähnten EA-Anweisungen.

---

1) Unter einem Block versteht man generell einen Sammelnamen für BEGIN-Blöcke und Prozedurblöcke, d.h. Statementfolgen, die durch Kopfvereinbarungen mit einem der beiden Schlüsselwörter BEGIN bzw. PROCEDURE (oder abgekürzt PROC) eingeleitet werden.

Zwischen den Gültigkeitsbereichsattributen EXTERNAL und INTERNAL einerseits und den Speicherklassenattributen AUTOMATIC und STATIC andererseits bestehen folgende ursächlichen Zusammenhänge:

a) INTERNAL schließt AUTOMATIC ein.
b) EXTERNAL schließt STATIC ein.

Die Zuweisung von Anfangswerten ist wie folgt geregelt:

a) Variablen mit dem Attribut AUTOMATIC werden die Anfangswerte beim Betreten des Blockes zugewiesen.

b) Variablen mit dem Attribut STATIC werden die Anfangswerte beim Laden des Programmes zugewiesen.

Es ist wohl einleuchtend, daß Variablen mit dem Attribut BASED keine Anfangswerte zugewiesen werden können, da für sie ja bekanntlich kein Hauptspeicherplatz zugeordnet ist.

Für statische externe Größen (Attribute EXTERNAL und STATIC) wird vom Kompilierer ein statischer Programmabschnitt [1] für jeden Block generiert, in dem für sie Erklärungen vorliegen. Der Binder [2] akzeptiert und verarbeitet jedoch nur das erste Erscheinen einer solchen Größe; jedes Folgeerscheinen wird grundsätzlich ignoriert. Wenn also Anfangswerte externen Variablen zugeordnet werden sollen, so muß dafür Sorge getragen werden, daß die richtigen Werte in der entscheidenden Erklärung zugewiesen werden.

In der Abb. 13.3. wird ein einfaches Programm vorgestellt, das von den verschiedenen Speicherklassen Gebrauch macht.

```
/**********************************************/
/**********************************************/
/**********************************************/
AAA:  PROCEDURE OPTIONS(MAIN);

        /* FALL 1 */
        /* ====== */

        DCL   (X AUTOMATIC INIT(111),
               Y STATIC INIT(222),
               Z EXTERNAL INIT(333))
                 DECIMAL FIXED(3);

        J=1;
        PUT PAGE LIST('ERGEBNISSE DES FALLES 1');
  L1:   PUT LIST(X,Y,Z) SKIP;

  BBA:  BEGIN;
        DCL   1 W EXTERNAL,
                2 WA DECIMAL FIXED(3) INIT(444),
                2 X DECIMAL FIXED(3) INIT(555);
        WA=Y;    X=Z;    Y=888;
  L2:   PUT LIST(X,Y,Z) SKIP;
        END BBA;

  L3:   PUT LIST(X,Y,Z) SKIP;
```

1) Im angloamerikanischen Sprachraum als „Control Section" bezeichnet.
2) Im angloamerikanischen Sprachraum als „Linkage Editor" bezeichnet.

```
BBB:   BEGIN;
       DCL   1 W EXTERNAL,
               2 VK DECIMAL FIXED(3) INIT(666),
               2 VB DECIMAL FIXED(3) INIT(777);
       Y=VB;    X=999;    Z=VK;    VB=0;
       END BBB;

 L4:   PUT LIST(X,Y,Z) SKIP;
       J=J-1;
       IF  J=0  THEN GOTO BBA;

    END AAA;

/*************************************************/
/*************************************************/


AAA: PROCEDURE OPTIONS(MAIN);

       /* FALL 2 */
       /* ====== */

       DCL  (X AUTOMATIC INIT(111),
             Y STATIC INIT(222),
             Z EXTERNAL INIT(333))
               DECIMAL FIXED(3);

       J=1;
       PUT PAGE LIST('ERGEBNISSE DES FALLES 2');
 L1:   PUT LIST(X,Y,Z) SKIP;

 BBA:  BEGIN;
       DCL   1 W STATIC INTERNAL,
               2 WA DECIMAL FIXED(3) INIT(444),
               2 X DECIMAL FIXED(3) INIT(555);
       WA=Y;    X=Z;    Y=888;
 L2:   PUT LIST(X,Y,Z) SKIP;
       END BBA;

 L3:   PUT LIST(X,Y,Z) SKIP;

 BBB:  BEGIN;
       DCL   1 W STATIC INTERNAL,
               2 VK DECIMAL FIXED(3) INIT(666),
               2 VB DECIMAL FIXED(3) INIT(777);
       Y=VB;    X=999;    Z=VK;    VB=0;
       END BBB;

 L4:   PUT LIST(X,Y,Z) SKIP;
       J=J-1;
       IF  J=0  THEN GOTO BBA;

    END AAA;

/*************************************************/
/*************************************************/
```

```
AAA: PROCEDURE OPTIONS(MAIN);

       /* FALL 3 */
       /* ====== */

       DCL  (X AUTOMATIC INIT(111),
             Y STATIC INIT(222),
             Z EXTERNAL INIT(333))
               DECIMAL FIXED(3);

       J=1;
       PUT PAGE LIST('ERGEBNISSE DES FALLES 3');
 L1:   PUT LIST(X,Y,Z) SKIP;

BBA:   BEGIN;
       DCL   1 W AUTOMATIC,
               2 WA DECIMAL FIXED(3) INIT(444),
               2 X DECIMAL FIXED(3) INIT(555);
       WA=Y;   X=Z;   Y=888;
 L2:   PUT LIST(X,Y,Z) SKIP;
       END BBA;

 L3:   PUT LIST(X,Y,Z) SKIP;

BBB:   BEGIN;
       DCL   1 W AUTOMATIC,
               2 VK DECIMAL FIXED(3) INIT(666),
               2 VB DECIMAL FIXED(3) INIT(777);
       Y=VB;   X=999;   Z=VK;   VB=0;
       END BBB;

 L4:   PUT LIST(X,Y,Z) SKIP;
       J=J-1;
       IF   J=0   THEN GOTO BBA;

     END AAA;


/*************************************************/
/*************************************************/
/*************************************************/
```

Abb. 13.3. Speicherklassenattribute in einem Beispielprogramm

Die Abb. 13.4. zeigt die Ausgabe des Programmes, das in Abb. 13.3. aufgelistet ist. Außerdem werden zwei weitere Ausgaben präsentiert, die sich ergeben, wenn einmal die Erklärungen der Variablen W in BBA und BBB in STATIC INTERNAL geändert wurden, und zum anderen, wenn die beiden Variablen W als AUTOMATIC gelten. Da den Variablen mit dem Attribut AUTOMATIC der Speicherplatz erst beim Betreten eines Blockes zugeordnet wird, schließen sich die beiden Attribute AUTOMATIC und EXTERNAL gegenseitig aus. Der Gebrauch von AUTOMATIC erzwingt das Attribut INTERNAL, der von EXTERNAL das Attribut STATIC.

```
ERGEBNISSE DES FALLES 1
   111                222                333
   333                888                333
   111                888                333
   999                333                222
   222                888                222
   999                888                222
   999                222                333

ERGEBNISSE DES FALLES 2
   111                222                333
   333                888                333
   111                888                333
   999                777                666
   666                888                666
   999                888                666
   999                  0                666

ERGEBNISSE DES FALLES 3
   111                222                333
   333                888                333
   111                888                333
   999                777                666
   666                888                666
   999                888                666
   999                777                666
```

Abb. 13.4. Ausgabe des Beispielprogrammes von Abb. 13.3. sowie Ausgaben zweier Variablen des Beispielprogrammes von Abb. 13.3.

Zur Abb. 13.3. wäre noch folgendes zu bemerken: Die Speicherklassenattribute für die in AAA erklärten Variablen sind effektlos, da diese Prozedur nur ein einziges Mal betreten wird und die Prozedur auch nicht intern zu einem anderen Block ist. Die in BBA zugewiesenen Anfangswerte bleiben ohne Belang, da nachfolgende Ergibtanweisungen diese Werte verändern. Gelangen wir nach BBB mit dem Attribut EXTERNAL für die Hauptstruktur W, so entspricht VK dem Strukturelement WA und VB dem Element X. Das kommt daher, daß die Struktur selbst extern ist und nicht geändert werden kann; die einzige zulässige Änderung für W besteht in der Namensgebung für die Strukturelemente. Wir müssen außerdem sicherstellen, daß die Struktur sich immer dann in gleicher Weise zusammensetzt, wenn wir eine auf sie bezogene Erklärung in

unser Programm aufnehmen müssen. Beachten wir stets, daß wir nur die internen Namen der Strukturelemente ändern dürfen.

Wenn die Erklärungen mit den Attributen STATIC und INTERNAL vorgenommen werden, so wird beim ersten Aktivieren von BBB dem Strukturelement VB der Wert 000 zugewiesen. Beim zweiten Betreten von BBB kommt es zur Ausführung der Ergibtanweisung

```
Y = VB ;
```

wodurch Y der Wert 000 zugewiesen wird. Ist dagegen W als extern deklariert, so wird in BBA dem Strukturelement X der Wert von Z zugewiesen, der zuvor in BBB gesetzt wurde. Bei Erklärung von W als INTERNAL haben die beiden Strukturvariablen W nichts miteinander zu tun. Schließlich wäre noch zu bemerken, daß bei Zugehörigkeit von W zur automatischen Speicherklasse die Resultate des zweiten Durchlaufens von BBB die gleichen sind, die beim ersten Durchlaufen erzielt werden; das hat seine Ursache in der Tatsache, daß die Anfangswerte bei jedem Durchlaufen von BBB erneut zugeordnet werden und nicht nur beim ersten Mal.

Die Benutzung des Attributes STATIC kann u. U. die Wirksamkeit der meisten Programme geringfügig verbessern.

13.9. Funktionen

Neben den in PL/1 eingefügten Funktionen kann ein Programmierer auch seine eigenen Funktionen schreiben und aufrufen. Von der IBM her wird außerdem eine Menge von Funktionen als SSP [1] zur Verfügung gestellt. In dieser Menge ist eine ganze Reihe gelegentlich recht nützlicher mathematischer und statistischer Funktionen enthalten. Eine Funktion ist in PL/1 als eine besondere Art von Prozeduren abzufassen, nämlich als sogenannte Funktionsprozedur. In der Abb. 13.5. ist die Form dargestellt, wie eine Funktion aufgerufen wird, aber auch die Form, wie die Funktion selbst zu schreiben ist. Die Funktion VOKALE dient dazu, die Anzahl der in einer Zeichenkette vorhandenen Vokale zu bestimmen, wobei die Länge der Kette von Fall zu Fall verschieden sein kann.

Das Hauptprogramm ist dadurch eindeutig bestimmt, daß OPTIONS(MAIN) im PROCEDURE-Statement steht. Damit ist dem Kompilierer mitgeteilt, daß ABC die Prozedur ist, der zunächst die Steuerung zu übergeben ist, wenn der Ablauf des Objektprogrammes eingeleitet wird. Im Hauptprogramm beziehen wir uns auf den Namen VOKALE; wir verwenden ihn als Funktionsnamen. Da dieser Name ein externer Name ist, muß er sich den Regeln für die Betriebssysteme einordnen und sollte deshalb aus nicht mehr als sechs (DOS) bzw. sieben Zeichen (OS) bestehen.

Um die Funktion VOKALE im Hauptprogramm aufzurufen, genügt es also, einfach ihren Namen niederzuschreiben und ihre Argumente, falls vorhanden, in Klammern zu setzen. Ist mehr als ein Argument beim Aufruf mitzugeben, so sind diese durch Kommas voneinander zu trennen; mit anderen Worten: es ist eine Argumentenliste zu bilden. Die Argumente selbst dürfen durchaus als Ausdrücke niedergeschrieben werden und dabei selbstverständlich in sich auch andere Funktionen einschließen. Für die Argumente besteht die einzige Einschränkung, daß sie mit ihren Attributen den entsprechenden Parametern angepaßt sein müssen, die bei der Funktionsdefinition eingeführt wurden.

1) SSP ist die Abkürzung von Scientific Subroutine Package.

```
ABC:  PROCEDURE OPTIONS(MAIN) ;
      /* HAUPTPROGRAMM --- PROGRAMMIERT VON HORST ELMAR */
      DECLARE  ARY CHARACTER(32),
               ASU CHARACTER(32),
               NUM BINARY FIXED(15),
               JK DECIMAL FIXED(3) ;
      DECLARE  VOKALE RETURNS (BINARY FIXED(31)) ;
      .
      .
      .
      JK = 32 ;
      NUM = VOKALE(ARY,JK) ;
      .
      .
      .
      NUM = VOKALE(ASU,Ø17) ;
      .
      .
      .
    END ABC ;
```

```
VOKALE:  PROCEDURE(ZK,LAENGE) RETURNS(BINARY FIXED(31)) ;
         /* FUNKTIONSPROZEDUR --- PROGRAMMIERT VON HORST ELMAR */
         DECLARE  ZK CHARACTER(32),
                  LAENGE DECIMAL FIXED(3) ;
         DECLARE  ZAEHL BINARY FIXED(31),
                  B CHARACTER(1) ;
         ZAEHL = Ø ;
         DO  J=1 TO LAENGE BY 1 ;
             B = SUBSTR(ZK,J,1) ;
             IF  B='A' | B='E' | B='I' | B='O' | B='U'
                   THEN  ZAEHL = ZAEHL + 1 ;
         END ;
         RETURN(ZAEHL) ;
       END VOKALE ;
```

Abb. 13.5. Funktionsprozedur VOKALE und ihr Aufruf in einem Hauptprogramm

Im Hauptprogramm haben wir außerdem den Funktionsnamen mit dem Attribut

RETURNS(datenattribute)

erklärt. Damit ist gesagt, wie die Attribute des Funktionswertes aussehen. Das RETURNS-Attribut kann entfallen, wenn der Funktionsname [1] mit einem der Buchstaben I bis N beginnt und der Funktionswert die Attribute

REAL BINARY FIXED(15)

1) Man beachte jedoch, daß bei Vorhandensein einer DEFAULT-Vereinbarung andere Voraussetzungen vorliegen können.

besitzt bzw. der Funktionsname mit einem anderen Buchstaben anfängt und der Funktionswert die Attribute

```
REAL DECIMAL FLOAT(6)
```

aufweist.

Hinsichtlich der Abfassung der Funktionsprozedur selbst stellen wir einige markante Unterschiede gegenüber den bisher niedergeschriebenen Prozeduren fest. Die Option

```
OPTIONS(MAIN)
```

fehlt, da es sich nicht um ein Hauptprogramm handelt. Andererseits bemerken wir das Erscheinen einer neuen Option, nämlich der Option

```
RETURNS(datenattribute)
```

Diese zeigt den gleichen Aufbau wie das entsprechende Attribut bei der Erklärung des Funktionsnamens im Hauptprogramm. Sie kann unter den gleichen Bedingungen weggelassen werden, die wir bei der Besprechung des Attributes RETURNS kennengelernt haben. Auch durch die RETURNS-Option wird nämlich nichts anderes festgelegt als die Attribute des Funktionswertes.

Letztlich begegnen wir dem RETURN-Statement in einer uns bisher unbekannten Form; es liegt nämlich in der Form

```
RETURN(ZAEHL) ;
```

vor.

Diese Form der RETURN-Anweisung kann nur bei Funktionsprozeduren verwendet werden. Der Wert des Ausdruckes, der in die Klammern eingeschlossen ist, wird als Funktionswert an die aufrufende Prozedur zurückgegeben. Allgemein lautet dieses Statement:

```
RETURN(ausdruck) ;
```

Wenn diese Anweisung ausgeführt wird, so wird der Wert von ausdruck so umgewandelt, daß er die Attribute des Funktionswertes besitzt, die ja bekanntlich entweder in der RETURNS-Option festgelegt wurden oder die bei deren Fehlen standardmäßig zugeordnet werden. Der so entstandene Wert wird dann als Funktionswert ans aufrufende Programm zurückgegeben. Die RETURN-Anweisung ist bei Funktionsprozeduren unbedingt erforderlich, sie kann keineswegs durch ein END-Statement ersetzt werden.

Die PROCEDURE-Vereinbarung besitzt folgende allgemeine Form:

```
name: PROCEDURE(p1,p2,···,pn) [RETURNS(attribute)] ;
```

Hierbei bedeuten die p_i die Parameter der Prozedur. Fehlt die Option RETURNS, so werden dem Funktionswert die üblichen Standardattribute zugeordnet, d.h. beginnt der Funktionsname mit I, J, K, L, M oder N, so besitzt der Funktionswert die Attribute REAL BINARY FIXED(15), beginnt er mit irgendeinem andren Buchstaben, so werden ihm die Attribute REAL DECIMAL FLOAT(6) zugeordnet [1]. Es ist nicht unbedingt erforderlich, daß die Attribute der Argumente im aufrufenden Programm mit den Attributen der Parameter in der Funktionsprozedur übereinstimmen, doch müssen dann, wenn das nicht der Fall ist, vom Kompilierer gesonderte Instruktionen generiert werden, die die erforderlichen Umwandlungen ablaufen lassen.

1) Eine DEFAULT-Vereinbarung kann diese Regeln außer Kraft setzen.

Wir sagten bisher, daß die Attribute der beim Aufruf an die Funktionsprozedur übergebenen Argumente mit den Attributen der Parameter übereinstimmen müssen. Diese Einschränkung können wir fallen lassen, wenn wir uns des ENTRY-Attributes bei der Erklärung des Funktionsnamens in der aufrufenden Prozedur bedienen. Durch dieses Attribut können wir der aufrufenden Prozedur nämlich sagen, welche Attribute die einzelnen Parameter der Funktionsprozedur besitzen; es tritt somit konsequent neben das RETURNS-Attribut, durch das ja bekanntlich der aufrufenden Prozedur gesagt wurde, welche Attribute der zurückgegebene Funktionswert mitbringt. Liegt das ENTRY-Attribut vor, so brauchen wir beim Aufruf nicht unbedingt dafür Sorge zu tragen, daß die an die Funktionsprozedur übergebenen Argumente die gleichen Attribute besitzen wie die in der Funktionsprozedur verwendeten Parameter. Dadurch, daß deren Attribute dem Kompilierer über das ENTRY-Attribut mitgeteilt werden,ist er nämlich in der Lage, Instruktionen zu erzeugen, die die Argumente noch vor dem Aufruf so umwandeln, daß sie an die Stelle der Parameter treten können. Das ENTRY-Attribut besitzt allgemein die Form:

```
ENTRY(attribute [,attribute]···
```

Durch attribute ist jeweils festgelegt, wie die Attribute des Parameters aussehen, der an der entsprechenden Stelle der Parameterliste in der Prozedurvereinbarung steht. Fehlt das ENTRY-Attribut, so wird unterstellt, daß die Parameter die gleichen Attribute wie die beim Aufruf mitgegebenen Argumente haben. Bei unserem Beispiel von Abb. 13.5. hätten wir somit in der aufrufenden Prozedur, hier also im Hauptprogramm, auch schreiben können:

```
DCL VOKALE  RETURNS (BINARY FIXED(31))
            ENTRY (CHARACTER(32), DECIMAL FIXED(3)) ;
```

Wegen der Tatsache, daß die Attribute der Argumente von Haus aus mit denen der Parameter übereinstimmen, konnte das ENTRY-Attribut weggelassen werden.

In Funktionsprozeduren können die Dimensionsgrenzen von Bereichen auch durch Ausdrücke festgelegt werden; es ist nicht mehr nötig, sie durch Konstanten zu umreißen. Die Ausdrücke werden beim Aufruf der Funktion ausgerechnet. Ein solches Vorgehen schließt natürlich ein, daß beim Aufruf der Funktion Informationen an sie übergeben werden, die es ihr gestatten, die Dimensionsgrenzen zu ermitteln.

Bei unserem Beispiel (Abb. 13.5.) schrieben wir die Funktion so, daß sie eine vom Rest des Programmes vollständig getrennte Prozedur bildete. Es ist jedoch durchaus erlaubt, daß eine Prozedur von einer anderen Prozedur umgeben wird, d.h. in einer anderen Prozedur enthalten ist. Das Hineinstellen der Funktionsprozedur ins Hauptprogramm würde dann zu einer Programmgestaltung führen,wie sie in Abb. 13.6. dargestellt ist.

```
ABC:  PROCEDURE OPTIONS(MAIN) ;
        :
        :
      VOKALE:  PROCEDURE(ZK,LAENGE)  RETURNS (BINARY FIXED(31)) ;
                 :
                 :
               END VOKALE ;
        :
        :
      END ABC ;
```

Abb. 13.6. Funktionsprozedur VOKALE als interne Prozedur

Eine interne Prozedur nimmt also innerhalb eines Programmes praktisch die gleiche Stellung ein wie ein BEGIN-Block. Eine solche interne Prozedur kann jedoch nicht linear betreten werden; in dieser Hinsicht unterscheidet sie sich grundlegend von den BEGIN-Blöcken und den DO-Gruppen. Halten wir ein für allemal fest, daß eine Funktionsprozedur nur durch einen Aufruf aktiviert werden kann, niemals durch einen sequentiell verfolgten Weg. Wird beim linearen Ablauf eines Programmes eine interne Funktion durch das Erscheinen einer Prozedurvereinbarung gefunden, so umgeht das Objektprogramm die Prozedur und fährt einfach mit der Anweisung fort, die auf das END-Statement folgt, das zur Prozedurvereinbarung gehört. Ist eine Prozedur in einer anderen Prozedur enthalten, so gelten für die Gültigkeitsbereiche von Namen die gleichen Regeln wie wir sie für BEGIN-Blöcke kennengelernt haben (siehe Abschnitt 13.7.).

Ist eine Prozedur in einer anderen Prozedur enthalten, so darf der Funktionsname nicht in der aufrufenden Prozedur erklärt werden. Auf unser Beispiel von Abb. 13.6. bezogen, heißt das, daß im Hauptprogramm eine Vereinbarung der Form

```
DCL  VOKALE RETURNS(···) ;
```

fehlen muß, die wir bei externen Prozeduren (siehe Abb. 13.5.) als unbedingt notwendig erkannt haben.

Es ist möglich, eine interne Funktionsprozedur mittels eines GOTO-Statements zu verlassen. Die Verzweigung muß jedoch zu einer Anweisung erfolgen, die innerhalb der Prozedur liegt, in deren Innern die Funktionsprozedur selbst liegt. In diesem Fall wird kein Funktionswert berechnet. Ein solches anormales Verlassen (anormaler Ausgang) einer Prozedur sollte normalerweise nicht vorgesehen werden.

Im allgemeinen wird man eine Prozedur, wenn sie in einer anderen enthalten sein soll, an das Programmende verbannen. Dann kann auch ein END-Statement zum Abschließen beider Prozeduren verwendet werden, sofern es sich auf die Eingangsstelle der übergeordneten Prozedur bezieht.

Als Parameter einer Funktion dürfen irgendwelche skalaren Variablen, Bereichsvariablen oder Strukturvariablen verwendet werden.

Funktionen dürfen sich auch selbst aufrufen. Soll eine Funktionsprozedur eine solche rekursive Eigenschaft besitzen, so muß in der Kopfvereinbarung der Prozedur die Option RECURSIVE notiert werden. Als Beispiel für eine rekursive Prozedur wollen wir die Berechnung von Fakultäten betrachten (Abb. 13.7.). Bekanntlich ist

$$n! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot n$$

Nach Voraussetzung soll bei unserem Programmbeispiel $n \geq 1$ sein.

```
FAKUL:  PROCEDURE(N) RECURSIVE
                     RETURNS (BINARY FIXED(31)) ;
        /* PROGRAMMIERT VON HORST ELMAR --- 13.Ø5.1966 */
        /* DER FUNKTIONSWERT IST GLEICH N FAKULTAET */
        DECLARE  N BINARY FIXED(31) ;
        IF  N=1  THEN RETURN(1) ;
                 ELSE RETURN(N*FAKUL(N-1)) ;
      END FAKUL ;
```

Abb. 13.7. Rekursive Funktionsprozedur zur Berechnung der Fakultät n!

Hinsichtlich der Speicherplatzzuordnung bei rekursiven Prozeduren ist folgendes zu bemerken: Sind Variablen mit dem Attribut AUTOMATIC bereits Speicherplätze zugeordnet, weil die Prozedur schon aufgerufen wurde, so werden bei einem erneuten Aufruf der Prozedur diese Speicherplätze zurückgestellt; es wird vielmehr eine weitere Menge von Speicherplätzen den Variablen zugeordnet. Diese neue Menge wird in dem Augenblick wieder freigegeben, wenn die Prozedur zum ersten Mal beendet wird. Sofort anschließend wird die zurückgestellte Speicherplatzmenge wieder hervorgeholt und mit ihr gearbeitet, bis eine erneute Beendigung der Prozedur auch diese Speicherplätze wieder freigibt. Dieser Vorgang ist notwendig, wenn rekursive Prozeduren zur Programmierung zugelassen werden sollen.

Ein weiteres Beispiel für eine rekursive Funktionsprozedur finden wir in der Aufgabengruppe 35, Aufgabe 12.

PL/1-Programme dürfen selbstverständlich auch Funktionen aufrufen, die in anderen Sprachen, beispielsweise in FORTRAN oder in der Assemblersprache geschrieben sind.

Um bestimmte Aufgaben mit Hilfe von Funktionsprozeduren zu lösen, gibt es mehrere Gründe:

a) die Funktion wird für mehrere Programme benötigt und man will sie nur ein einziges Mal programmieren,
b) das Problem ist so verwickelt, daß die Fehlersuche im Programm wesentlich vereinfacht werden kann, wenn es möglich ist, die Funktion getrennt auf Fehler zu durchsuchen.

13.10. Subroutinen

Im letzten Abschnitt behandelten wir eine Funktion, mit deren Hilfe dem aufrufenden Programm ein Wert übergeben werden konnte, der die Anzahl der Vokale in einer 32-stelligen Zeichenkette widerspiegelte. Nehmen wir nunmehr an, daß wir sowohl die Anzahl der Vokale als auch die Anzahl der Leerzeichen in der Zeichenkette feststellen wollen. Eine wesentliche Eigenschaft einer Funktion zeigt sich aber darin, daß sie dem aufrufenden Programm einen und nur einen Wert, den sogenannten Funktionswert, übergeben kann. Wenn dagegen mehr als ein Wert zurückgegeben werden muß, dann muß eine sogenannte Subroutine geschrieben werden. Wir wollen eine solche demonstrieren, indem wir im wesentlichen das Programm aus dem letzten Abschnitt wiederholen, doch soll die Bestimmung der Zahl der Vokale ergänzt werden durch die Bestimmung der Zahl der Leerzeichen. Die Subroutine selbst soll mit VOKLEE bezeichnet werden; die Zahl der Vokale soll in ihr unter Z2, die Zahl der Leerzeichen unter Z1 bestimmt werden (Abb. 13.8.).

Rein äußerlich gesehen ist die Subroutine VOKLEE einer Funktion sehr ähnlich. Hier ist sie als unabhängige (externe) Prozedur niedergeschrieben, sie hätte aber ebensogut auch als interne Prozedur in die Hauptprozedur hineingestellt werden können, so wie wir es bei BEGIN-Blöcken und Funktionsprozeduren bereits kennengelernt haben. Die Subroutine-Prozedur VOKLEE weist vier Parameter auf. Vom aufrufenden Programm her werden Werte für die Größen ZK und JK übernommen, an die aufrufende Prozedur zurückgegeben werden Werte an die Argumente, die den Parametern Z1 (Zähler für Leerzeichen) und Z2 (Zähler für Vokale) entsprechen. Die soeben getroffene Feststellung über die Art des Werteaustausches ist praktisch ohne Belang; es können durchaus auch Parameter vorgesehen werden, die sowohl Werte erwarten als auch bei der Rückkehr wieder Werte abgeben.

```
ABC:  PROCEDURE OPTIONS(MAIN) ;
        /* HAUPTPROGRAMM */
        DECLARE  (ARY,ASU) CHARACTER(32),
                 (LZAEHL,VZAEHL) BINARY FIXED(15) INIT(Ø) ;
        DECLARE  JK DECIMAL FIXED(3) ;
        DECLARE  VOKLEE ENTRY(CHAR(32),DEC FIXED(3),,BIN FIXED(15)) ;
        :
        CALL VOKLEE(ARY,JK,LZAEHL,VZAEHL) ;
        NUM = 32 - LZAEHL - VZAEHL ;
        :
        CALL VOKLEE(ASU,Ø17,KL,KV) ;
        :
      END ABC ;
```

```
VOKLEE:  PROCEDURE(ZK,JK,Z1,Z2) ;
           /* SUBROUTINE-UNTERPROGRAMM --- (HORST ELMAR) */
           DECLARE  ZK CHARACTER(32),
                    JK DECIMAL FIXED(3),
                    (Z1,Z2) BINARY FIXED(15),
                    B CHARACTER(1) ;
           Z1,Z2 = Ø ;
           DO  J=1 TO JK ;
               B = SUBSTR(ZK,J,1) ;
               IF  B='A' | B='E' | B='I' | B='O' | B='U'
                   THEN  Z2 = Z2 + 1 ;
                   ELSE  IF  B=' '  THEN  Z1 = Z1 + 1 ;
           END ;
         END VOKLEE ;
```

Abb. 13.8. Subroutineprozedur VOKLEE und ihr Aufruf in einem Hauptprogramm

Die Parameter können in einer beliebigen Reihenfolge niedergeschrieben werden; allein wichtig ist nur, daß die beim Aufruf der Subroutine-Prozedur verwendeten Argumente in der gleichen Reihenfolge erscheinen wie die Parameter. Da alle Bezeichnungen in der Subroutine-Prozedur als interne Bezeichnungen erklärt sind, d.h. das Attribut INTERNAL besitzen, haben die Variablen JK des Hauptprogrammes und JK des Unterprogrammes nichts miteinander zu tun.

Die CALL-Anweisung im Hauptprogramm veranlaßt, daß die augenblicklichen Werte der Argumente an die Parameter der Subroutine übergeben werden. Wenn die Programmablaufsteuerung auf die RETURN-Anweisung oder auf das END-Statement stößt, so werden die jetzt vorhandenen Parameterwerte den Argumenten im Hauptprogramm

zur Verfügung gestellt. In Abb. 13.8. ist auf die RETURN-Anweisung verzichtet worden; es besteht auch kein Grund dazu, sie zu verwenden, da das physische Ende des Unterprogrammes VOKLEE (END-Statement) mit dem logischen Ausgang zusammenfällt.

Da eine Subroutine nicht einen Einzelwert zurückgibt, ist im Hauptprogramm der Name der Subroutine ohne RETURNS-Attribut, dagegen mit dem ENTRY-Attribut erklärt worden. Das Erscheinen dieses Attributes sagt bekanntlich dem Kompilierer, daß der in der Erklärung enthaltene Name der Name einer externen Prozedur ist. Im ENTRY-Attribut haben wir nur die Attribute des ersten, zweiten und vierten Parameters von VOKLEE beschrieben. Wir hätten ohne weiteres auch auf diese Beschreibungen verzichten können, da die Attribute der Parameter exakt mit denen der Argumente übereinstimmen. Wenn ein Subroutinename nicht in einem DCL-Statement in Erscheinung tritt, so bedeutet seine Verwendung in einer CALL-Anweisung, daß er implizit das Attribut ENTRY zugeordnet erhält. Subroutinenamen werden deshalb gewöhnlich nicht in DCL-Vereinbarungen eingeschlossen.

Wir haben beim Schreiben von Subroutinen auf zweierlei zu achten:

a) Die Option RETURNS darf in der Prozedurvereinbarung nicht erscheinen.
b) Die RETURN-Anweisung muß frei von einem Ausdruck sein.

Beim Aufruf von Subroutinen haben wir die Anweisung

CALL name(a_1,a_2,$\cdots$,a_n) ;

zu benutzen. Wenn beim Aufruf keine Argumente a_1, a_2,$\cdots$,a_n erforderlich sind, d.h. für die Subroutine keine Parameter erklärt sind, so genügt die Anweisung

```
CALL name ;
```

In beiden Fällen bedeutet name den Namen der Subroutine. Der Name einer externen Prozedur ist von Haus aus selbstverständlich ein externer Name. Im Gegensatz zur Funktionsprozedur sind jedoch an diesen Namen keine Datenattribute geknüpft.

Subroutinen werden im Gegensatz zu Funktionen oft in die aufrufende Prozedur hineingestellt, also als interne Prozedur eingesetzt. Dadurch ist es möglich, die Subroutine aufzurufen, ohne Argumente angeben zu müssen. Haben wir ein umfangreiches Programm zu schreiben, so können wir sehr viel Zeit damit einsparen, daß wir das Programm in einzelne Teile zerlegen. Einer dieser Abschnitte, das Haupt- oder Wurzelsegment kann dann einfach einen Teil nach dem anderen aufrufen, dienst also allein dazu, die Verbindung zwischen den einzelnen Teilen herzustellen. Der Vorteil ist hauptsächlich darin zu suchen, daß wir dann jeden Programmteil unabhängig von den anderen als Subroutine schreiben und danach testen können, bevor wir sie in das Gesamtprogramm einfügen.

Würden wir VOKLEE als interne Subroutine-Prozedur in das Hauptprogramm einfügen (siehe Abb. 13.8.), so könnten wir etwa so codieren, wie es die Abb. 13.9. zeigt.

```
ZK = ARY ;
CALL VOKLEE ;
NUM = 32 - Z1 - Z2 ;
```

Abb. 13.9. VOKLEE als interne Prozedur (siehe auch Abb. 13.8.)

Wir müßten dann allerdings die Erklärungen von ZK, Z1 und Z2 in das Hauptprogramm verlegen und auf die Erklärung von JK im Unterprogramm gänzlich verzichten. Außerdem wäre die Parameterliste in der Prozedurvereinbarung von VOKLEE zu entfernen.

Funktionen und Subroutinen werden gemeinsam als Unterprogramme bezeichnet. Dieser Oberbegriff charakterisiert also Programmteile, die von einem anderen übergeordneten Programm aufgerufen werden und die normalerweise nach ihrer Beendigung zum aufrufenden Programm zurückkehren. Jedes Unterprogramm kann selbst wieder andere Unterprogramme aufrufen (bei mathematischen Anwendungen ist es nicht ungewöhnlich, wenn bis zu 15 und mehr ineinander verschachtelte Unterprogrammaufrufe vorliegen).

Wie bei einer Funktion kann auch bei Subroutinen ein GOTO-Statement als anormaler Ausgang aus der Prozedur verwendet werden, doch müssen die gleichen Bedingungen beachtet werden, die schon bei den Funktionen aufgeführt wurden.

Um noch einmal die Betrachtungen dieses und des vorigen Abschnittes zusammenzufassen, muß man sagen, daß in PL/1 eigentlich nur folgende Unterscheidung zwischen einer Funktion und einer Subroutine getroffen wird: Eine Funktion kann direkt mit ihrem Namen aufgerufen werden, ein CALL-Statement ist nicht erforderlich; außerdem wird von einer Funktion an das aufrufende Programm ein Wert, der sogenannte Funktionswert, zurückgegeben. Liegt eine interne Funktion vor, d.h. ist ihr Text vollständig in die aufrufende Prozedur eingeschlossen, so kann die Verwendung von Variablen, die außerhalb der Funktion definiert sind, durchaus genau so effektvoll gehandhabt werden wie bei Subroutinen. Somit ist also auch bei ihnen wie bei den Subroutinen eine Wertänderung von derartigen Variablen möglich. Es ist deshalb durchaus vernünftig, wenn man die Funktionen nur als eine besondere Art von Subroutinen betrachtet, nämlich als solche, die zusätzlich die Fähigkeit besitzen, daß man sie implizit aufrufen kann.

13.11. ENTRY-Statements

Sehr häufig sind zwei verschiedene Berechnungen einander so ähnlich, daß es für den Programmierer sinnvoll erscheint, sie in einem einzigen Programm zusammenzuschließen. So erfordern z.B. die beiden eingefügten Funktionen zur Berechnung der Sinuswerte für Winkel im Bogenmaß und im Gradmaß nur eine zusätzliche Multiplikation zu Beginn des Programmes. In gleicher Weise unterscheiden sich die Rechengänge zur Berechnung des Arcustangens im Grad- bzw. im Bogenmaß nur durch eine einzige Multiplikation am Schluß des Programmes. Sowohl bei Subroutinen als auch bei Funktionen kann deshalb das ENTRY-Statement dazu verwendet werden, um zusätzliche Eingangspunkte (Eingangsstellen) für eine Prozedur festzulegen. Es besitzt folgende allgemeine Form:

name: ENTRY [(p [,p] ···)] [RETURNS(datenattribute) ;

Dem ENTRY-Statement muß wie beim PROCEDURE-Statement mindestens ein Name vorausgehen; wie üblich gilt dieser als Eingangsname. Die durch ENTRY-Statements festgelegten Eingangspunkte in eine Prozedur bezeichnet man als sekundäre Eingangspunkte, das PROCEDURE-Statement hingegen markiert den primären Eingangspunkt. Die Option RETURNS ist wie üblich nur für Funktionsprozeduren maßgebend. Im großen und ganzen gesehen sieht also das ENTRY-Statement genau so aus wie das PROCEDURE-Statement; die Option OPTIONS fehlt freilich.

Die in einem ENTRY-Statement angegebenen Parameter müssen nicht mit denen übereinstimmen, die im PROCEDURE-Statement stehen. Genau wie beim PROCEDURE-Statement ist es auch beim ENTRY-Statement nicht erforderlich, daß überhaupt Parameter spezifiziert werden. Wird eine Prozedur mit dem Namen aufgerufen, der bei einem ENTRY-Statement steht, so beginnt der Ablauf der Prozedur mit der ersten Anweisung, die auf die ENTRY-Vereinbarung folgt. Wird beim Ablauf einer Prozedur ein ENTRY-Statement erreicht, so bleibt dieses unbeachtet.

Das Unterprogramm von Abb. 13.10. bestimmt den Index des größten oder des kleinsten Elementes von einem aus zehn Elementen bestehenden Vektor.

```
MAXIND:  PROCEDURE(A) RETURNS(BINARY FIXED(15)) ;
           DECLARE  (A(1Ø),X) DECIMAL FLOAT(6) ;
           X = MAX(A(1),A(2),A(3),A(4),A(5),A(6),
                   A(7),A(8),A(9),A(1Ø)) ;
       V:  DO  I=1 TO 1Ø ;
               IF  X=A(I)  THEN RETURN(I) ;
           END ;
MININD:  ENTRY(A) RETURNS(BINARY FIXED(15)) ;
           X = MIN(A(1),A(2),A(3),A(4),A(5),A(6),
                   A(7),A(8),A(9),A(1Ø)) ;
           GOTO V ;
         END MAXIND ;
```

Abb. 13.10. Beispiel für eine Funktionsprozedur mit einer sekundären Eingangsstelle

Erscheint im Hauptprogramm die Anweisung

```
J = MAXIND(R) ;
```

und ist R im Hauptprogramm als Bereich mit zehn Elementen in Gleitpunktdarstellung erklärt, so wird J auf den Index gesetzt, durch den das größte Element von R gekennzeichnet ist. Schreiben wir im Hauptprogramm

```
J = MININD(R) ;
```

so wird K auf den Index des kleinsten Elementes von R gesetzt. Wir sollten auch unsere Aufmerksamkeit darauf richten, daß unser Funktionsunterprogramm selbst wieder zwei Funktionen aufruft, nämlich die eingefügten Funktionen MAX und MIN. Es gibt keine Beschränkungen hinsichtlich des Aufrufes von Funktionen innerhalb von Funktionen.

13.12. Spezielle Testhilfen

Zum Austesten von Programmen kann ein spezielles Subroutine-Unterprogramm aufgerufen werden. Hierzu ist das Statement

```
CALL PLIDUMP[('kennung' [,ausdruck])] ;
```

an geeigneter Stelle in ein Programm einzufügen. Unter ausdruck kann eine Zeichenkette oder eine dezimale Konstante niedergeschrieben werden; dieser Wert dient ausschließlich nur zur Identifizierung der Ausgabe. Die Routine PLIDUMP veranlaßt das Drucken gewisser Informationen, beim Betriebssystem DOS über SYSLST, beim Betriebssystem OS über die Datei PLIDUMP, für die eine Datenmenge- bzw. Gerätezuordnung mittels des DD-Steuerstatements der Jobsteuersprache zu treffen ist. Die Kennbuchstaben, die unter kennung codiert ist, zeigen an, in welcher Form die Ausgabe vorgenommen werden soll. Die zulässige Kenninformationen sind in Abb. 13.11. aufgeführt.

Erfolgt der Aufruf von PLIDUMP ohne Kenninformationen, so wird im Betriebssystem OS die Ausgabe standardmäßig mit der Kennung TFCNHNB vorgenommen, im Betriebssystem DOS ebenfalls mit der Kennung TFCNHNB, es sei denn, daß die Standardsystemmaßnahme beim Auftreten der ERROR-Bedingung abläuft. Für diese ist nämlich TFCHB

Kenninformation	Bedeutung der Kenninformation bzw. Art der Ausgabe
T	Ausgabe von Informationen über alle aktiven Blöcke, wodurch eine Verfolgung des Programmablaufes möglich ist
NT	Unterdrückung der Informationen über alle aktiven Blöcke
F	Ausgabe einer Übersicht über die Attribute und die Pufferinhalte aller eröffneten Dateien
NF	Unterdrückung der Dateienübersicht
S	Abbruch des Programmes nach der Ausgabe ohne Auftreten der FINISH-Bedingung
C	Fortführung des Programmablaufes nach erfolgter Ausgabe
H	Ausgabe des Inhalts aller vom Programm belegten Hauptspeicherstellen in sedezimaler Form
NH	Keine Ausgabe des Inhalts der vom Programm belegten Hauptspeicherstellen
B	Ausgabe des Inhalts der vom Programm benutzten Steuerblöcke in sedezimaler Form
NB	Keine Ausgabe des Inhaltes der vom Programm benutzten Steuerblöcke

Abb. 13.11. Art der durch den Aufruf von PLIDUMP veranlaßten Druckausgabe

als Standard unterstellt. Ob bei einer im Betriebssystem OS aufgetretenen ERROR-Bedingung die Standardsystemmaßnahme die Routine PLIDUMP aufruft, hängt von der Entscheidung des Systemprogrammierers ab.

Die Subroutine PLIDUMP ist sicher eines der wertvollsten Werkzeuge beim Austesten von Programmen.

Rückkehrcode in sedezimaler Form	Ursache
0000	Normale Beendigung des Programmes
1000	Beendigung eines Programmes infolge der Ausführung der folgenden Anweisungen: a) STOP b) EXIT c) CALL PLIDUMP
2000	Abbruch eines Programmes infolge des Auftretens der ERROR-Bedingung, ohne daß dabei eine auf die Bedingungen ERROR oder FINISH bezogene Oneinheit existiert, die in einem GOTO-Statement mündet
4000 bis 4024	Auftreten von Fehlern, die auf den PL/1-Kompilierer oder auf Bibliotheksunterprogramme zurückzuführen sind, bzw. Feststellung eines Programmierfehlers in einer anderen Programmiersprache, in der ein vom PL/1-Programm aufgerufenes Unterprogramm geschrieben ist. Der Rückkehrcode 4012 besagt speziell, daß für das PL/1-Programm nicht genügend Hauptspeicherplatz zur Verfügung steht.

Abb. 13.12. Rückkehrcodes

Im Betriebssystem OS steht eine weitere nützliche Subroutine zur Verfügung. Das Unterprogramm PLIRETC kann mit einem Argument aufgerufen werden, dessen Attribute gleich BINARY FIXED(31) sind und dessen Wert im Bereich von 1 bis 999 einschließlich liegt. Durch diese Subroutine wird ein Rückkehrcode gesetzt, der durch den COND-Parameter in nachfolgenden EXEC-Statements der Jobsteuersprache abgefragt werden kann, soweit diese zum gleichen Job gehören. Von einem PL/1-Programm selbst werden standardmäßig die in der Abb. 13.12. aufgeführten Rückkehrcodes gesetzt.

Der durch PLIRETC gesetzte Rückkehrcode wird zu dem vom System gesetzten Rückkehrcode addiert, es sei denn, daß ein Code oberhalb 4000 (sed) vorgegeben wird.

Beim Austesten eines Programmes kann durchaus auch die eingefügte Funktion ONCODE von Nutzen sein. Diese Funktion gibt einen Funktionswert zurück, der zur Identifizierung der Fehlerursache herangezogen werden kann. Hierzu sind die in den PL/1-Handbüchern enthaltenen Tabellen zu Rate zu ziehen.

Aufgabengruppe 34

1) In einem Programm ist nach Feststellen des Dateiendes der Eingabedatei EIN der Wert der Endsumme ESU und die Anzahl der Summanden zu drucken sowie eine Abschlußmeldung über die Konsole zu schreiben. Deute die Programmierung an, wobei für die Verarbeitung nach dem Dat eiende ein BEGIN-Block eingesetzt werden soll.

2) Zur Berechnung der Kubikwurzeln für Radikanden in Gleitpunktdarstellung ist ein Unterprogramm zu schreiben, dem das in Abschnitt 12.10. vorgestellte Rechenverfahren zugrundeliegt.

3) Füge das Unterprogramm von Aufgabe 2) in ein Programm ein, das eine Wertetafel mit folgenden Werten druckt:

$$n,\ n^2,\ n^3;\qquad \sqrt{n},\qquad \sqrt{10n},\qquad \sqrt[3]{n},\qquad \sqrt[3]{10n},\qquad \sqrt[3]{100n}$$

Hierbei soll n die Werte von 1000 bis 9999 durchlaufen.

4) Aufgabe 3) ist so abzuändern, daß jeweils eine Leerzeile verbleibt, wenn die Werte für n = 1005, 1010, 1015 usw. gedruckt sind. Jeweils fünfzig Werte von n sind auf eine Seite zu drucken.

5) Der Gegenwartswert S einer Kapitalanlage von 1.- DM ist gegeben durch

$$S = \frac{(1+r)^n - 1}{r}$$

Hierbei bedeutet r den Zinsfuß und n ist gleich der Zahl der abgelaufenen Zinsperioden. Eines der Probleme bei Geldanlagen besteht aber gerade darin, den Zinsfuß zu bestimmen, wenn S und n bekannt sind. Bei der Berechnung von r kann man so vorgehen, daß man für r zunächst zwei Näherungswerte r_1 und r_2 so aufsucht, daß $S_1 < S < S_2$ ist. S_1 und S_2 findet man, wenn man r_1 bzw. r_2 in die Gleichung für S einsetzt.

Nach Ermittlung von r_1 und r_2 bestimmt man einen neuen Näherungswert für r durch Einsetzen der Werte von S, S_1, S_2, r_1 und r_2 in die Interpolationsgleichung

$$\frac{S - S_1}{S_2 - S_1} = \frac{r_i - r_1}{r_2 - r_1}$$

Der neue Näherungswert ist mit r_i bezeichnet. Nach der Bestimmung von r_i errechnet man S_i durch Einsetzen von r_i in die für S angegebene Formel und ersetzt dann S_1 oder S_2 durch S_i, je nachdem, ob $S_i<S$ oder ob $S_i>S$ ist. Nach diesem Austausch wird erneut ein neuer Näherungswert für r_i errechnet, daraus ein neuer Wert für S_i bestimmt usw. Dieses Iterationsverfahren wird solange fortgesetzt, bis ein Wert für r_i gefunden ist, dessen zugehöriger Wert S_i genügend nahe bei S liegt.

Schreibe ein Programm, mit dessen Hilfe r bis auf sechs Dezimalstellen genau berechnet werden kann. Zur Eingabe gelangen Werte für n und S; ausgegeben werden sollen die Werte von n, S und r.

Zur Berechnung der Funktion S(r) soll ein Funktionsunterprogramm eingesetzt werden.

6) Die Anzahl C(n,k) der Kombinationen von k Elementen aus einer Menge von n Elementen (Kombinationen k-ter Ordnung) ist gegeben durch die Beziehung

$$C(n,k) = \frac{n!}{k!\,(n-k)!}$$

Dabei ist

$$0! = 1$$
$$x! = x\cdot(x-1)! = x\cdot(x-1)\cdot(x-2)\cdot\cdots\cdot 3\cdot 2\cdot 1$$

Schreibe ein Funktionsunterprogramm zur Berechnung von C(n,k). Die Werte von n und k sind als ganze Dualzahlen vorgegeben.

7) Die kumulative binomische Wahrscheinlichkeit ist durch die Formel

$$B(k;n,p) = \sum_{r=0}^{k} C(n,r)p^r(1-p)^{n-r}$$

gegeben.

Schreibe ein Programm, das aus folgenden Schritten besteht:

a) Einlesen der Werte von n und p
b) Drucken einer Tabelle der Werte von B(k;n,p) für $k=0,1,2,\cdots,n$.

Zur Berechnung der Werte von B(k;n,p) ist das in Aufgabe 6) entwickelte Funktionsunterprogramm einzusetzen.

8) Beim Bridgespiel befinden sich in der eigenen Hand und in der des Partners gerade 26 Karten. Die verbleibenden 26 Karten sind auf die zwei Gegenspieler verteilt. Wenn die beiden Gegenspieler zusammen k Pikkarten haben, so beträgt die Wahrscheinlichkeit, daß ein Gegenspieler $n\leq k$ Pikkarten hat, gerade

$$w = C(k,n)\cdot C(26-k,13-n) + C(26,13)$$

Schreibe ein Programm, das eine Tabelle der Wahrscheinlichkeiten (auf vier Dezimalstellen genau) für $n=1,2,3,\cdots,13$ ausgibt.

9) Wenn m eine negative ganze Zahl ist, so ist die kombinatorische Funktion gegeben durch

$$C(-n,k) = \frac{(-n)(-n-1)\cdots(-n-r+1)}{k(k-1)\cdots 2\cdot 1}$$

Hierbei sind n und k positive ganze Zahlen, für die $k>n$ gilt. Schreibe ein Subroutine-Unterprogramm, das für alle ganzzahligen Werte von m den Funktionswert C(m,k) berechnet.

10) Die Wahrscheinlichkeit, mit einem Wurf von n Würfeln die Summe von j·n Augen zu erzielen, ist durch

$$P(j,n)=6^{-n}\sum_{S=0}^{\infty}(-1)^{j+7s}C(n,s)C(-n,j-6s)$$

gegeben. Schreibe ein Programm, das die Wahrscheinlichkeit zur Erzielung der möglichen Gesamtaugenzahlen bei einem Wurf von n Würfeln errechnet.

11) Angenommen, es ist eine Matrix mit den Elementen

$$a_{ij} \text{ mit } i=1,2,3,\cdots,m \text{ und } j=1,2,3,\cdots,n$$

gegeben; außerdem sind zwei Vektoren mit den Elementen

$$b_i \text{ mit } i=1,2,3,\cdots,m \text{ bzw.}$$
$$c_j \text{ mit } j=1,2,3,\cdots,n$$

bekannt.

Das Grundproblem von Linear Programming besteht darin, einen Vektor x_j so zu bestimmen, daß

I. $\sum_{j=1}^{n} a_{ij}x_j \quad b_i$ für $i=1,2,\cdots,m$

und

II. $x_j \geq 0$ für $j=1,2,\cdots,n$

ist und daß außerdem

III. $z=\sum_{j=1}^{n} c_jx_j$

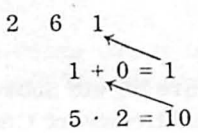
so groß wie möglich ist.

Schreibe ein PL/1-Programm zur Bestimmung der allgemeinen Lösung für n=2 und m≤7. Die Werte für die Elemente der Matrix $\mathcal{A}$ bzw. der Vektoren $\mathcal{B}$ und $\mathcal{L}$ sind über Lochkarten einzulesen. Die Ausgabe der x-Werte und des z-Wertes soll auf Lochkarten erfolgen. Es ist anzunehmen, daß alle Eingabewerte größer als oder gleich Null sind.

12) Gewisse IBM-Maschinen, wie z.B. die Locher der Gerätetypen IBM 29 und IBM 129 sind in der Lage, Prüfziffern zu stanzen oder zu überprüfen, die aufgrund des Modulo-10-Prüfverfahrens der IBM entstehen. Eine solche Prüfziffer wird wie folgt gebildet, dargestellt am Beispiel der Zahl 123752:

a) Jede zweite Ziffer, beginnend bei der Ziffer höchster Ordnung, wird ausgesondert:
1 3 5

b) Die ausgesonderten Ziffern werden jeweils mit 2 multipliziert. Ist ein Produkt größer als 9, so sind die beiden Ziffern des Produktes zu addieren. Dieser Schritt führt auf



c) Alle im Schritt b) erhaltenen Ziffern werden addiert:

2 + 6 + 1 = 9

d) Alle nicht ausgesonderten Ziffern der ursprünglichen Zahl werden addiert:

2 + 7 + 2 = 11

e) Addition der in den Schritten c) und d) erhaltenen Teilsummen:

9 + 11 = 20

Ist die so entstandene Summe durch 10 teilbar, so genügt die Zahl dem Modulo-10-Prüfverfahren der IBM.

Schreibe ein Unterprogramm, das die Prüfziffer nach diesem Verfahren ermittelt. Dem Unterprogramm wird eine achtstellige Zeichenkette übergeben. An das aufrufende Programm ist eine gleichlange Kette zurückzugeben, dessen rechtsbündige Ziffer durch die korrekte Ziffer ersetzt ist, so daß dem Test nach dem Modulo-10-Verfahren Genüge geleistet ist.

13) Schließe das Ergebnis der Aufgabe 12) in ein Programm ein, das achtstellige numerische Felder liest (Spalten 9 bis 16 der Lochkarten eines Kartenpaketes). Diese sind auf gültige Prüfziffern zu untersuchen. Wenn das Kartenfeld irgendein nichtnumerisches Zeichen enthält, so soll eine Lochkarte gestanzt werden, die identisch der Eingabekarte ist. In allen anderen Fällen ist eine Karte zu stanzen, die den Eingabewert auf den Spalten 9 bis 16 und den nach dem Modulo-10-Verfahren gültigen Wert auf den Spalten 24 bis 31 enthält.

13.13. Neudefinitionen

Es kommt häufig vor, daß man sich auf eine Variable beziehen will oder muß, die man mit mehreren verschiedenen Namen bezeichnet hat. So ist es durchaus nicht unwahrscheinlich, daß mehrere Programmierer, die an einem Programm gearbeitet haben, für die gleiche Variable verschiedene Namen eingeführt haben. Vielleicht ist damit unter Umständen sogar ein gewisser Vorteil verbunden. Mitunter wollen wir aber auch zwei Variablen in verschiedenen Programmteilen den gleichen Speicherplatz zuordnen. Doch in jedem dieser beiden Fälle können wir die gleiche Neudefinition von Variablen durchführen, die wir jetzt besprechen wollen.

Beim einfachen Neudefinieren wird einfach einer bereits definierten Variablen ein neuer Name gegeben. Ein Beispiel möge das erläutern:

```
DCL   FSEGMENT CHAR(14),
      HAUSZU CHAR(14) DEFINED FSEGMENT ;
```

FSEGMENT und HAUSZU sind zwei verschiedene Namen für die gleiche Variable. Aber nicht nur für skalare Variablen können Neudefinitionen erfolgen, sondern auch auf Bereiche und Strukturen. Bei

```
DCL   A(5) FIXED(4),
      B(5) FIXED(4) DEFINED A ;
```

liegt eine Neudefinition vor: Dem unter dem Namen A bekannten Bereich wird der neue Name B gegeben. Das Attribut, das Neudefinitionen bewerkstelligt, heißt DEFINED oder abgekürzt DEF.

Bei Bereichen ist zweierlei möglich: Man kann einmal einem bereits bekannten Bereich einen neuen Namen zuordnen, der ebenfalls für den gesamten Bereich gilt; andererseits kann man aber auch nur einem Teil des ursprünglichen Bereiches einen neuen Namen geben. Im ersten Fall können wir z.B. schreiben:

```
DCL  A(9,9) FIXED(6) INIT(81)Ø),
     B(9,9) FIXED(6) DEFINED(A) ;
```

A und B werden als Namen für den gleichen Bereich angesehen. A heißt hierbei Basisname. Bei Festlegung eines neuen Namens wird also dieser dadurch eingeführt, daß ihm das Attribut DEFINED folgt, dem sich selbst wieder der Basisname anschließt. Anfangswerte (INITIAL-Attribut) können nur bei der Festlegung des Basisnamens zugewiesen werden, nicht jedoch in der Neudefinition. Im übrigen muß selbstverständlich der neue Name mit den gleichen Attributen erklärt werden, die bei der Erklärung des Basisnamens aufgeführt wurden. Eine Ausnahme bildet hier lediglich das Dimensionsattribut. Die Dimensionsgrenzen des neu definierten Bereiches B dürfen kleiner sein als die des Basisbereiches A. Somit ist z.B. die Vereinbarung

```
DCL  A(9,9) FIXED(6) INIT((81)Ø),
     C(7,9) FIXED(6) DEFINED(A) ;
```

ebenfalls gültig. Hier entspricht C nur den ersten sieben Zeilen von A. Wie wir bei den bisherigen Beispielen gesehen haben, darf die Basisvariable im Anschluß an das DEFINED-Attribut sowohl in Klammern als auch ohne Klammern geschrieben werden.

Bei Bereichen können wir auch ein spezielles Symbol der Form iSUB für Neudefinitionen benutzen. Hierbei ist i=1,2,···,n und bedeutet den 1.,2.,···,n-ten Index der Basisvariablen. Diese Art des Neudefinierens soll an einem Beispiel verfolgt werden.

```
DCL  A(9,9) FIXED(6) INITIAL((81)Ø),
     D(9) FIXED(6) DEFINED A(1SUB,1Ø-1SUB) ;
```

A ist ein zweidimensionaler Bereich mit 9·9 Elementen, denen sämtlich der Anfangswert Ø zugewiesen ist. D ist dagegen ein eindimensionaler Bereich, der aus neun Elementen mit den Indizes 1 bis 9 besteht. Seine Elemente entsprechen denjenigen Bereichselementen von A, die sinnbildlich ausgedrückt, von rechts oben nach links unten gehen. Es besteht damit folgende Namensidentität:

```
D(J) = A(J,1Ø-J)
```

Das Symbol iSUB muß mindestens einmal irgendwo in der Neudefinition erscheinen, weil nur durch dieses Symbol die teilweise Zuordnung eines neuen Namens möglich ist. Die Ausdrücke, die die iSUB-Angaben enthalten, dürfen keine Multiplikation und Division von zwei iSUB-Angaben enthalten. Der Ausdruck 3SUB+4*5SUB ist also erlaubt, der Ausdruck 3SUB*5SUB dagegen nicht. Mit anderen Worten, die in den Dimensionsangaben der Neudefinition auftretenden Ausdrücke müssen von der Form

$$a_1 * 1SUB + a_2 * 2SUB + \cdots$$

sein. Hierbei dürfen die a_i selbst wieder beliebige Ausdrücke sein, die jedoch frei von iSUB-Angaben sind.

Die überlagernde Neudefinition von Ketten sieht im Grunde genommen vor, daß zwei oder mehr verschiedene Typen von Variablen den gleichen Speicherplatz oder Teile desselben belegen, der einer anderen Variablen, der Basisvariablen, zugeordnet ist. An einem einfachen Beispiel ist das leicht zu erkennen:

```
DCL   D CHAR(11),
      E CHAR(8) DEFINED D ;
```

Hier belegt die Variable E acht Bytes, wobei diese die gleichen sind wie die ersten acht Bytes der Basisvariablen D. Zum Vergleich sei noch einmal erwähnt, daß die korrespondierende oder einfache Neudefinition identische Attribute für die korrespondierenden Variablen erfordert, die überlagernde Neudefinition dagegen nicht.

Wollen wir z.B. erreichen, daß E die rechten acht Bytes von D belegt, so müssen wir schreiben:

```
DCL   1 SCHEIN,
        2 G CHAR(3),
        2 E CHAR(8),
      D CHAR(11) DEFINED SCHEIN ;
```

Bei dieser Neudefinition haben wir jedoch eine andere Basisvariable wählen müssen; zweckmäßiger ist es deshalb, wenn wir anstatt dessen das spezielle hierfür geschaffene Attribut

```
POSITION(d)
```

verwenden. Es legt fest, daß die Überlagerung erst auf der d-ten Stelle der Basisvariablen beginnen soll. Unter Benutzung dieses Attributes können wir dann einfach schreiben:

```
DCL   D CHAR(11),
      E CHAR(8) DEFINED D POSITION(4) ;
```

Die Zeichenkettenvariable E mit der Länge 8 entspricht also dem Teil der Zeichenkettenvariablen mit der Länge 11, die auf ihrer 4. Stelle beginnt.

Bei einer derartigen Neudefinition ist es nicht erlaubt, daß die neu definierte Variable einen Speicherplatz beansprucht, der über das Ende des Speicherplatzes hinausreicht, der der Basisvariablen zugewiesen wird. Ebenso darf der neu definierten Variablen kein Anfangswert zugewiesen werden.

Weiterhin ist es notwendig, daß die Attribute der neudefinierten Variablen mit den Attributen der Basisvariablen verträglich sind. Demzufolge sind nur folgende Arten des überlagernden Definierens erlaubt:

a) Überlagerndes Definieren bei Bitklassendaten
   Bitketten; Bereiche von Bitketten; Strukturen mit Bitketten

b) Überlagerndes Definieren bei Zeichenklassendaten
   Zeichenketten; numerische Variablen in Feldform (PICTURE-Attribut); Bereiche und Strukturen, die nur diese beiden Datentypen enthalten.

Bei dieser Art des Neudefinierens können iSUB-Angaben nicht eingesetzt werden.

Für alle Neudefinitionen gelten bestimmte Regeln. Anfangswerte können nur den Basisvariablen zugewiesen werden, den neudefinierten Variablen jedoch nicht. Jede neu definierte Variable wird mit dem Attribut INTERNAL versehen, sogar wenn die Basisvariable durch das Attribut EXTERNAL ausgezeichnet ist. Die Basisvariable, auf die eine Neudefinition bezug nimmt, darf weder durch das Attribut BASED noch durch das Attribut DEFINED gekennzeichnet sein. Hinsichtlich der neu definierten Variablen wäre noch zu bemerken, daß sie entweder die Stufennummer 1 oder keine Stufennummer besitzen müssen. Außerdem darf der von ihnen beanspruchte Hauptspeicherplatz nicht über das Ende des Hauptspeicherplatzes hinausragen, der von der Basisvariablen beansprucht wird.

13.14. Die Anordnung der Elemente in Strukturen

Die Verarbeitungsleistung eines Computers wird herabgemindert, wenn Daten gewisser Typen nicht auf geeignete Hauptspeichergrenzen ausgerichtet sind. Dem Programmierer stehen zwei Attribute,nämlich ALIGNED und UNALIGNED, zur Verfügung, mit denen er die Ausrichtung der von den Variablen benötigten Hauptspeicherplätze auf zweckmäßige Hauptspeicherunterteilungen regeln kann. Besonders gilt das selbstverständlich für Datengruppierungen, wie Bereiche und Strukturen.

Es gibt zwei Hauptgründe, Daten nicht auf passende Hauptspeicherunterteilungen auszurichten. Der erste Grund besteht darin, daß eine erhebliche Anzahl von Hauptspeicherplätzen eingespart werden kann, wenn umfangreiche Bereiche in die Verfahren verwickelt sind. So werden z. B. durch den Bereich BK insgesamt 1000 Hauptspeicherstellen belegt, wenn die Vereinbarung

```
DCL  BK(1ØØØ) BIT(1) ALIGNED ;
```

vorliegt. Jedes einzelne Element von BK beginnt nämlich infolge der durch ALIGNED verlangten Ausrichtung gerade ein Byte. Wird durch Verwendung des Attributes UNALIGNED auf die Ausrichtung verzichtet, so finden acht Elemente des Bereiches BK in einem Byte Platz, so daß für BK insgesamt nur 125 Bytes benötigt werden. Die Einsparung beträgt also 875 Bytes. Der zweite Grund, von der Ausrichtung bzw. Nichtausrichtung Gebrauch zu machen, ist darin zu suchen, daß man die Verträglichkeit zu Dateien anstreben muß, die entweder von in anderen Sprachen geschriebenen Programmen erzeugt wurden oder die zur Verarbeitung in Programmen vorgesehen werden müssen. So erzeugen z.B. in FORTRAN geschriebene Programme Dateien, bei denen die in den Sätzen enthaltenen Elemente stets auf Hauptspeicherunterteilungen ausgerichtet sind.

Wenn der Programmierer die Länge einer in die satzweise EA einbezogenen Struktur bestimmen will bzw. muß, so muß er unbenutzte Bytes (Füllbytes) bei seiner Längenberechnung berücksichtigen. Bei Verwendung der Anweisungen READ (mit der SET-Option) und LOCATE liegt es in der Verantwortlichkeit des Programmierers selbst, sicherzustellen, daß alle Datenelemente in geeigneter Weise ausgerichtet sind, es sei denn, er benutzt das Attribut UNALIGNED. Das Betriebssystem selbst sorgt dafür, daß der erste Satz eines Blockes stets auf Doppelwortgrenze beginnt.

Die Angabe COBOL kann bekanntlich bei Dateierklärungen in das ENVIRONMENT-Attribut eingefügt werden. Auf solche Dateien dürfen nur die Anweisungen READ mit der INTO-Option und WRITE mit der FROM-Option bezogen werden. Bei Angabe von COBOL erzeugt der Kompilierer Instruktionen, die die Eingabe vom Format einer durch ein COBOL-Programm aufgezeichneten Struktur in das Format einer PL/1-Struktur überführen. Beide Strukturformate stimmen nämlich dann und nur dann überein, wenn das erste Datenbyte auf Doppelwortgrenze beginnt, keine Füllbytes vorhanden sind und außerdem alle Datenelemente ausgerichtet sind.

In der Abb. 13.13. sind die Längen und die Ansprüche, die an die Ausrichtung gestellt sind, für alle Datentypen aufgeführt, einschließlich einiger, die wir noch nicht besprochen haben. Die Regeln für die Anordnung der Datenelemente in Strukturen sind schlicht und einfach Regeln, die aussagen, wo und in welcher Anzahl in Strukturen Füllbits und Füllbytes eingeschoben werden. Wird allen Datenelementen das Attribut UNALIGNED gegeben, so gibt es keine Füllbytes, jedoch werden u. U. Füllbits benötigt. Besitzen alle Datenelemente hingegen das Attribut ALIGNED, so können Füllbytes dadurch vermieden werden, indem man in der Struktur zuerst alle Elemente aufführt, die Doppel-

Attribute des Datenelementes	Länge des Datenelementes in Bytes	Hauptspeicherausrichtung beim Attribut	
		ALIGNED	UNALIGNED
a) *Datenelemente mit standardmäßiger Zuordnung des Attributes UNALIGNED*			
BIT(n)	CEIL(n/8)	C	B
CHARACTER(n)	n	C	C
PICTURE	s	C	C
BIT(n) VARYING	2+CEIL(n/8)	H	C
CHARACTER(n) VARYING	2+n	H	C
b) *Datenelemente mit standardmäßiger Zuordnung des Attributes ALIGNED*			
DECIMAL FIXED(p,q)	CEIL((p+1)/2)	C	C
BINARY FIXED(p_1,q)	2	H	C
BINARY FIXED(p_2,q)	4	W	C
DECIMAL FLOAT(p_3)	4	W	C
DECIMAL FLOAT(p_4)	8	D	C
DECIMAL FLOAT(p_5)	16	D	C
BINARY FLOAT(p_6)	4	W	C
BINARY FLOAT(p_7)	8	D	C
BINARY FLOAT(p_8)	16	D	C
LABEL	8	W	C
POINTER	4	W	C
OFFSET	4	W	C
FILE	4	W	C
ENTRY	8	W	C
c) *Datenelemente, die stets das Attribut ALIGNED besitzen*			
TASK	16	W	-
EVENT	32	W	-
AREA(n)	16+n	W	-

Abb. 13.13. Ausrichtung und Nichtausrichtung von Datenelementen

Zeichenerklärungen

s	Anzahl der Abbildungszeichen ohne V und K
p_1	1,2,···,15
p_2	16,17,···,31
p_3	1,2,···,6
p_4	7,8,···,16
p_5	17,18,···,33
p_6	1,2,···,21
p_7	22,23,···,53
p_8	54,55,···,109

B	Ausrichtung auf Bitstelle
C	Ausrichtung auf Bytegrenze
H	Ausrichtung auf Halbwortgrenze
W	Ausrichtung auf Wortgrenze
D	Ausrichtung auf Doppelwortgrenze

wörter belegen, dann alle, die Wörter belegen, dann alle, die Halbwörter belegen; an diese schließen sich diejenigen an, die Bytegrenzen erfordern. Zum Schluß kommen die Bitketten. Natürlich ist es nicht immer möglich, Datenelemente in solch einer Reihenfolge anzuordnen. Es ist durchaus denkbar, daß Strukturteile ausgerichtet, andere nicht ausgerichtet sind. Geben wir das Attribut ALIGNED oder das Attribut UNALIGNED dem Hauptstrukturnamen, so bedeutet das, daß dieses Attribut auf alle Strukturelemente bezogen wird.

Die Anordnung der Elemente innerhalb einer Struktur wollen wir an einem Beispiel betrachten. Die Vereinbarung der Struktur finden wir in Abb. 13.14.

```
DECLARE  1 STRU ALIGNED,
           2 C CHARACTER(5),
           2 D,
             3 E,
               4 F DECIMAL FLOAT(16),
               4 G BINARY FIXED(31),
             3 H DECIMAL FIXED(5),
           2 I CHARACTER(2),
           2 J,
             5 K,
               6 L BINARY FIXED(31),
               6 M(4) BINARY FIXED(31),
             5 N CHARACTER(4) ;
```

Abb. 13.14. Beispiel für eine Strukturvereinbarung

Logische Stufennummer	Strukturteil	Länge und Begrenzung	Bytebelegung (mod 8)	
			Anfang	Ende
1	STRU	-	-	-
2	C	5 | C	Ø	4
2	D	-	-	-
3	E	-	-	-
4	F	8 | D	Ø	7
4	G	4 | W	Ø	3
3	H	3 | C	Ø	2
2	I	2 | C	Ø	1
2	J	-	-	-
3	K	-	-	-
4	L	4 | W	Ø	3
4	M	16 | W	Ø	7
3	N	4 | C	Ø	3

Zeichenerklärungen:
C Bytegrenze
W Wortgrenze
D Doppelwortgrenze

Abb. 13.15. Übersicht über die Strukturteile der in Abb. 13.14. vereinbarten Struktur

Bei der Strukturerklärung von Abb. 13.14. sind die Stufennummern willkürlich vergeben worden. Wir wollen uns zunächst einmal eine Übersicht (Abb. 13.15.) herstellen, in der für alle Strukturteile die kleinstmögliche, d.h. die logische, Stufennummer aufgeführt ist. Weiterhin soll die Übersicht die Länge jedes Datenelementes in Bytes, seine Ausrichtungserfordernisse und die laufende Nummer des ersten und des letzten Bytes innerhalb des von ihm belegten Hauptspeicherplatzes enthalten. Dabei beginnt die Numerierung mit 0 und wird modulo 8 durchgezogen, d.h. es wird von 0 bis 7 gezählt, danach wird mit 0 fortgefahren. Unterstrukturen weisen selbstverständlich keine Längeneintragung auf.

Wir beginnen bei den Unterstrukturen mit der höchsten logischen Stufennummer. In diesem Fall sind es E und K mit der Stufennummer 3. Wir paaren die beiden ersten Elemente von E und erhalten

```
(F) ——— 8 ¦D | Ø ¦7
(G) ——— 4 ¦W | Ø ¦3
```

Wir versuchen nun, das zweite Element G so nach links zu verschieben, daß es sich möglichst nahe an das erste Element F anschließt. Da das Byte 0 das unmittelbar auf Byte 7 folgende Byte ist, liegen die beiden Elemente bereits unmittelbar nebeneinander, so daß eine Verschiebung unmöglich ist. Wir verknüpfen jetzt beide Elemente, d.h. wir addieren ihre Längen, suchen uns die schärfste Begrenzungsanforderung heraus und kombinieren ferner die Bytebelegungen hinsichtlich Anfang und Ende. Wir erhalten damit

```
(F) ——— 12¦ D| Ø¦ 3
```

Mit diesem Quadrupel können wir nun den Eintrag von E in der Übersicht von Abb. 13.15. ergänzen und gleichzeitig die Eintragungen von F und G löschen. In gleicher Weise gehen wir bei der Unterstruktur K vor. Wir erhalten zunächst

```
(L) ——— 4 ¦W | Ø ¦3
(M) ——— 16 ¦W | Ø ¦7
```

Wir erkennen hieraus, daß wir den Bereich M um vier Bytes nach links ziehen können, womit wir

```
(K) ——— 2Ø¦W| Ø ¦3
```

erhalten.

Nach Erledigung der Unterstrukturen mit der logischen Stufennummer 3 wenden wir uns den Unterstrukturen auf der Stufe 2 zu. Hier gibt es deren zwei, nämlich D und J. Für D erhalten wir

```
(E) ——— 12¦D | Ø ¦3
(H) ——— 3 ¦C | Ø ¦2
```

Das Element H kann um vier Bytes nach links verschoben werden , so daß sich

```
(D) ——— 15¦D  Ø  6
```

ergibt. In gleicher Weise verfahren wir mit J, wodurch wir aus

```
(K) ——— 2Ø¦W| Ø ¦3
(N) ———  4 ¦C| Ø ¦4
```

schließlich

```
(J) ——— 24 ¦W | Ø ¦7
```

erhalten.

Unsere Übersicht haben wir damit erheblich reduziert. Sehen wir uns zunächst einmal das erzielte Zwischenergebnis in Abb. 13.16. an.

Logische Stufennummer	Strukturteil	Länge und Begrenzung	Bytebelegung (mod 8)	
			Anfang	Ende
1	STRU	-	-	-
2	C	5 C	ø	4
2	D	15 D	ø	6
2	I	2 C	ø	1
2	J	24 W	ø	7

Zeichenerklärungen:

C	Bytegrenze
W	Wortgrenze
D	Doppelwortgrenze

Abb. 13.16. Übersicht über die Strukturteile der in Abb. 13.14. vereinbarten Struktur nach Zusammenfassung der Strukturelemente der Unterstrukturen der höchsten logischen Stufennummer

Wir haben nun mehrere Strukturteile mit der gleichen logischen Stufennummer vor uns. Wir beginnen unsere Verschiebeüberlegungen mit den ersten beiden

(C) —— 5 ¦C| ø ¦4
(D) —— 15¦D| ø ¦ 6

Wegen der schärferen Begrenzungsforderung bei D müssen wir von der Linksverschiebung absehen. Um die Lücke zwischen C und D zu schließen, schieben wir C nach rechts; hieraus resultiert

(C) —— 5 ¦C| 3 ¦ 7
(D) —— 15¦D| ø ¦ 6

Zusammengefaßt folgt

(C-D) —— 2ø¦D | 3 ¦ 6

Dieser ersten Kombination fügen wir das dritte Element I hinzu.

(C-D) —— 2ø¦D | 3 ¦ 6
(I) —— 2¦C | ø ¦ 1

Das zweite Element kann um ein Byte nach links geschoben werden. Daraus folgt

(C-D-I) —— 22¦D |3 ¦ ø

Wir sind nunmehr so weit, um auch das vierte Element, nämlich J, anzuhängen.

(C-D-I) —— 22¦D |3 ¦ ø
(J) —— 24¦W |ø ¦ 7

Da J auf Wortgrenze auszurichten ist, ist nur eine Verschiebung von J um vier Bytes nach links möglich. Wir ordnen um und erhalten

(C-D-I) —— 22¦D | 3 ¦ ø
(J) —— 24¦W | 4 ¦ 3

Da die erste Strukturteilekombination auf Doppelwortgrenze beginnen muß, ist deren Verschiebung nach rechts nicht möglich. Somit ergibt sich eine Lücke von 3 Bytes zwischen (C-D-I) und (J). Wir fügen also drei Füllbytes ein und erhalten damit

```
(C-D-I) ——— 22 ¦ D ¦ 3 ¦ ø
(Füllbytes) ——— 3 ¦ C ¦ 1 ¦ 3
(J) ——— 24 ¦ W ¦ 4 ¦ 3
```

Zusammengefaßt ergibt sich

```
(STRU) ——— 49 ¦ D ¦ 3 ¦ 3
```

Daraus folgt, daß die Struktur STRU insgesamt 49 Bytes belegen wird. Wir gehen nunmehr zurück zur Übersicht von Abb. 13.14. und bestimmen den Abstand [1] jedes Strukturelementes vom Anfang von STRU an. Das Resultat unserer Bemühungen haben wir in Abb. 13.17. festgehalten. Wir bemerken hierzu, daß die Struktur nicht auf einer Doppelwortgrenze beginnt.

Logische Stufennummer	Strukturteil	Abstand vom Beginn der Struktur
1	STRU	ø
2	C	ø
2	D	5
3	E	5
4	F	5
4	G	13
3	H	17
2	I	2ø
2	J	25
3	K	25
4	L	25
4	M	29
3	N	45

Abb. 13.17. Abstand der Strukturteile vom Strukturbeginn (Beispiel von Abb. 13.14.)

Wenn die Struktur STRU als basisbezogene Variable in eine READ-Anweisung mit der SET-Option oder in eine LOCATE-Anweisung eingeht, so werden keine Füllbytes vor dem ersten Element von STRU eingefügt; die Struktur wird jedoch (wegen der Puffer) auf Doppelwortbegrenzung ausgelegt. Es ist Sache des Programmierers, ein Scheinelement von drei Bytes Länge vor C zu legen, damit die gewünschte Doppelwortbelegung für F erhalten bleibt.

Bei Sätzen im Format FB [2] ist es ebenfalls Sache des Programmierers, die Satzlänge durch 8 teilbar zu machen, so daß auch der 2. und die folgenden Sätze eines Blockes an einer Doppelwortgrenze beginnen. Bei Sätzen im Format VB [3] ist außerdem das vier Byte umfassende Präfixfeld vor jedem Satz zu berücksichtigen: Die um 4 vermehrte Satzlänge sollte deshalb ein Vielfaches von 8 sein.

Zum Schluß halten wir es für gerechtfertigt, wenn wir noch einmal darauf verweisen, daß selbstverständlich alle Strukturen ohne Füllbytes erklärt werden können. Probleme tauchen überhaupt nicht auf, wenn die erklärten Strukturen auf Doppelwortgrenze beginnen. Um das zu erreichen, ist es meist sinnvoll, die Strukturerklärungen um Scheinelemente zu bereichern.

1) Im angloamerikanischen Sprachraum als Offset bezeichnet.
2) Geblockte Sätze fester Länge.
3) Geblockte Sätze variabler Länge.

13.15. Datenumwandlungen

PL/1 gestattet es dem Programmierer, diejenigen Datentypen einzusetzen, die er für die Lösung der anstehenden Aufgabe als am passendsten erachtet. Kehrtwendend bringt diese Flexibilität jedoch eine andere Problematik auf ihn zu: Er muß sich ständig im klaren darüber sein, daß die daraus resultierenden Datenumwandlungen zeitaufwendig sind und somit den Ablauf eines Objektprogrammes verlangsamen. Die für die Datenumwandlungen geltenden Regeln sind im Anhang E zusammengefaßt. In diesem Abschnitt wollen wir uns mit einigen Beispielen begnügen, die uns zeigen sollen, was passieren kann, wenn Datenumwandlungen leichtfertig gehandhabt werden.

Beispiel 1

Bei der Festpunktdivision entsteht ein Resultat mit größtmöglicher Genauigkeit. Diese braucht nicht mit der Genauigkeit übereinzustimmen, die tatsächlich gewünscht wird (Abb. 13.18.).

```
DCL  (A,B) DECIMAL FIXED(5,3) ;
B = 32.112 ;
A = B + 1/3 ;
```

Abb. 13.18. Beispiel 1 für Datenumwandlungen

Betrachten wir die Vorgänge bei der letzten Anweisung. Zuerst wird die Division erledigt. Die beiden Konstanten sind mit den Attributen DECIMAL FIXED(1) behaftet. Der Quotient weist somit die Attribute DECIMAL FIXED(15,14) auf (siehe hierzu die entsprechende Regel im Anhang E). Somit ergibt sich aus 1/3 der Wert Ø.33333333333333. Addieren wir nun den Wert von B mit der Genauigkeit (5,3) auf den Quotienten mit seiner Genauigkeit (15,14), so bekommen wir eine Summe, die ebenfalls die Genauigkeit (15,14) aufweist. Die Auswertung des Ausdruckes auf der rechten Seite des Ergibtzeichens führt somit nur auf den Wert 2.44533333333333, obgleich das korrekte Resultat 32.445333333333333 ist. Nach Angleichung an die Genauigkeit (5,3), d.h. an die definierte Genauigkeit von A, wird A der Wert 2.445 zugewiesen. Befindet sich die Bedingung FIXEDOVERFLOW im wirksamen Status, so wird sie wegen des Verlustes der führenden 3 auf der Zehnerstelle auch auftreten. Würden wir dagegen die Ergibtanweisung

```
A = B + Ø.333 ;
```

niederschreiben, so ergibt sich bei Ausführung dieser Anweisung das einwandfreie Resultat 32.445 für A.

Beispiel 2

Die Benutzung von dezimalen Konstanten im Zusammenhang mit Ketten ist nicht nur höchst unwirtschaftlich, sondern führt öfters auch auf unliebsame Resultate (Abb. 13.19.).

```
DCL  C BIT(1) ;
DCL  D DECIMAL FIXED(2) ;

C = 1 ;
IF  C=1  THEN  D = 2Ø ;
         ELSE  D = 3Ø ;
```

Abb. 13.19. Beispiel 2 für Datenumwandlungen

Diese Statementfolge bringt uns wider Erwarten auf den Zweig, der D den Wert 3Ø zuweist. Die Ergibtanweisung soll eine Konstante mit den Attributen DECIMAL FIXED(1) einer Bitkettenvariablen der Länge 1 zuweisen. Damit eine solche Zuweisung überhaupt erfolgen kann, muß die Konstante zunächst in einen Wert mit den Attributen BINARY FIXED(4) und anschließend in eine Bitkette mit den Attributen BIT(4) umgeformt werden, was letztlich das Resultat 'ØØØ1'B erbringt. Diese Bitkette wiederum ist nunmehr auf die Attribute des Zuweisungszieles, also BIT(1) abzustimmen. Da die Zuweisung bei Bitketten von links nach rechts erfolgt, gehen die rechtsbündigen drei Bits verloren, und es verbleibt

```
C = 'Ø'B ;
```

Der nachfolgende Vergleichsausdruck in der IF-Anweisung wird dagegen unterschiedlich gehandhabt. Hier ist die 1 eine arithmetische Konstante; diese ist maßgebend dafür, daß ein algebraischer Vergleich eingeleitet wird. Also muß der Wert von C in die verschlüsselte arithmetische Form gebracht werden. Er wird als BINARY FIXED(1) angesehen, also gleich ØB. Die Konstante 1 wiederum wird zu einem Wert mit den Attributen BINARY FIXED(5), d.h. zu ØØØØ1B, umgeformt. Beide Werte sind nicht gleich, was logischerweise zum Betreten der ELSE-Klausel führt.

Beispiel 3

Arithmetische Operationen mit Bitketten sind wohl möglich, doch mit Vorsicht zu genießen (Abb. 13.20.).

```
DCL  E BIT(31) ;
DCL  F BIT(8) ;

F = 'ØØ11ØØØ1'B ;
F = F + 2 ;
```

Abb. 13.20. Beispiel 3 für Datenumwandlungen

Die Konstante 2 besitzt die Attribute DECIMAL FIXED(1), F ist eine Bitkette der Länge 8. Vor ihrer additiven Verknüpfung werden beide Werte in ein Datenelement umgewandelt, das die Attribute BINARY FIXED aufweist; die Konstante 2 erhält außerdem das Genauigkeitsattribut (2), der aus F hervorgehende Wert dagegen (31). Danach wird die arithmetische Operation ausgeführt, die das korrekte Resultat 51 (dezimal) mit den Attributen BINARY FIXED(31) hervorbringt. Bei der Zuweisung zur achtstelligen Bitkette F werden nur die acht Bits höchster Ordnung herangezogen, die restlichen Bits gehen verloren. Somit ergibt sich für F der Wert 'ØØØØØØØØ'B. Um das korrekte Resultat der Addition auch F wirklich zuzuweisen, bestehen folgende zwei Möglichkeiten der Codierung:

a)
```
F = BINARY(F+2,8,Ø) ;
```
Bei dieser Lösung wird die eingefügte Funktion BINARY aufgerufen, die für eine präzise Zuweisung die Vorbereitungen trifft.

b)
```
E = F + 2 ;
F = SUBSTR(E,24,8) ;
```

Beispiel 4

Wenn Dezimalbrüche in Dualbrüche transferiert werden, so entstehen Abbrechfehler, die zur Verfälschung von Resultaten führen können (Abb. 13.21.).

```
DCL  G BINARY FIXED(31,4),
     H DECIMAL FIXED(5,2) ;

H = 23.2Ø ;
G = H ;
H = G ;
```

Abb. 13.21. Beispiel 4 für Datenumwandlungen

Für G ergibt sich der Wert 23.1875, da der Bruch nach vier Dualstellen abgebrochen wird, für H dagegen der Wert 23.18 wegen des Abschneidens auf zwei Dezimalstellen.

Beispiel 5

Die Umwandlung von Zeichenketten führt auf unterschiedliche Werte, wenn diese in arithmetische Operationen eingehen. Die Ursache hierfür liegt in den Änderungen der Genauigkeit begründet (Abb. 13.22.).

```
DECLARE  I DECIMAL FIXED(5,2),
         J DECIMAL FLOAT(5),
         K CHARACTER(4) ;

K = '2.5Ø' ;
I = K ;          /* I WIRD DER WERT ØØ2.5Ø ZUGEWIESEN */
J = K ;          /* J WIRD DER WERT 2.5Ø ZUGEWIESEN */
I = K + K ;      /* I WIRD DER WERT ØØ4.ØØ ZUGEWIESEN */
```

Abb. 13.22. Beispiel 5 für Datenumwandlungen

Ist keine arithmetische Operation in eine Ergibtanweisung verwickelt, so wird die Zeichenkette in die Genauigkeit der empfangenden Variablen umgewandelt. Ist hingegen eine arithmetische Operation auszuführen, so wird die Zeichenkette in einen Wert mit den Attributen DECIMAL FIXED(15) umgewandelt, der in diesem Fall gleich 2 ist, wodurch es in der letzten Ergibtanweisung zur Zuweisung von 4 für die Variable I kommt.

Beispiel 6

Die Umwandlung eines arithmetischen Wertes in eine Zeichenkette erfolgt unter der Regie der listengesteuerten reihenweisen EA (Abb. 13.23.).

Wenn die letzte Ergibtanweisung

```
P = P + 2 ;
```

zur Ausführung gelangt, so wird zunächst der P zugewiesene Wert in einen Wert mit den Attributen DECIMAL FIXED(15) umgewandelt, auf den dann die durch die Attribute DECIMAL FIXED(1) ausgezeichnete Konstante 2 addiert wird. Die daraus resultierende Summe 5 besitzt selbst die Attribute DECIMAL FIXED(15). Aus diesem Wert muß vor der eigentlichen Zuweisung eine Zeichenkette geschaffen werden, was auf eine solche der Länge 18 führt (17 Leerzeichen gefolgt von der Ziffer 5). Von dieser Zeichenkette werden die vier linken Zeichen P zugewiesen, da P nur die Länge 4 aufweist. Alle anderen Zeichen gehen durch Abschneiden verloren.

```
DCL  L CHARACTER(7(,
     M DECIMAL FIXED(4,1),
     N DECIMAL FIXED(2) ;
M = -123.4 ;
N = 38 ;
L = M ;          /* L GLEICH 'b-123.4' */
L = N ;          /* L GLEICH 'bb38bb' */
DCL  P CHARACTER(4) ;
P = M ;          /* P GLEICH 'b-12' */
P = N ;          /* P GLEICH 'bbb3' */
P = 'bbb3' ;
P = P + 2 ;      /* P GLEICH 'bbbb' */
```

Anmerkung: b bedeutet hier das Leerzeichen

Abb. 13.23. Beispiel 6 für Datenumwandlungen

13.16. Ketten mit variabler Länge

Es ist in einem Unterprogramm möglich, eine Kettenvariable zu definieren, deren Länge selbst durch einen Parameter ausgedrückt wird (siehe hierzu das Beispiel in Abb. 13.24.).

```
MRN:  PROCEDURE (A,B,C) ;
        DCL  A CHAR(B),
             B BINARY FIXED(15),
             C BIT(B+8) ;
        .
        .
      END MRN ;
```

Abb. 13.24. Kettenlänge als Parameter

Beim Aufruf der Prozedur MRN werden B und B+8 errechnet und dann zur Längenbestimmung der zur automatischen Speicherklasse gehörenden Variablen A und C benutzt. Obgleich uns diese Methode in die Lage versetzt, unterschiedliche Längen bei verschiedenen Aufrufen der Prozedur einzurichten, sind die Längen innerhalb eines Aufrufs, d.h. während einer Prozedurausführung, fest.

PL/1 gestattet es jedoch, Kettenvariable festzulegen, deren Länge Änderungen selbst während des Programmablaufes unterliegen kann. Es ist dazu nur erforderlich, das Attribut VARYING zur Erklärung der Kette hinzuzufügen. Für basisbezogene Variable ist diese Möglichkeit nicht gegeben. Die erklärte Länge ist in einem solchen Fall die maximal mögliche Länge, die eine Kette annehmen kann. Bei einem Unterprogramm kann die maximale Länge durch einen Ausdruck festgehalten werden.

Speichermäßig wird eine Kette variabler Länge wie folgt behandelt: Zur Linken der von ihr besetzten Hauptspeicherplätze wird ein Halbwort angelegt, in dem die augenblickliche Länge steht. Wird einer Kettenvariablen mit variabler Länge ein Wert zugewiesen, so wird diesem Halbwort ebenfalls ein neuer Wert zugewiesen, der gleich der Länge der zugewiesenen Kette ist. Mittels der eingefügten Funktion LENGTH hat der Programmierer Zugriff zur augenblicklichen Länge einer Kette. Teile der Kette können mit Hilfe der Pseudovariablen SUBSTR geändert werden, eine Längenänderung findet dabei nicht statt. In Abb. 13.25. wird ein Beispiel demonstriert.

```
DCL  D CHARACTER(18) VARYING ;
DCL  E CHARACTER(5) INITIAL('ABCDE') ;

D = E ;                    /* D BESITZT DIE LAENGE 5 */
D = E || E || E || 'W' ;   /* D BESITZT DIE LAENGE 11 */
J = LENGTH(D) ;            /* J WEIST DEN WERT 11 AUF */
D = '' ;                   /* D BESITZT DIE LAENGE Ø */
```

Abb. 13.25. Beispiel mit einer Zeichenkette variabler Länge

Bei einer basisbezogenen Struktur sind, wie wir soeben feststellten, Ketten mit einer variablen Länge nicht angebbar. Es ist jedoch gestattet, eine basisbezogene Struktur zu erklären, in der Teile enthalten sind, deren Länge oder deren Dimensionsgrenze(n) durch Variable außerhalb der Struktur festgelegt sind. Die Zuweisung der aktuellen Werte erfolgt in dem Augenblick, in dem der betreffenden Struktur Speicherplatz zugeordnet wird. Doch sehen wir uns hierzu zunächst einmal das Beispiel in Abb. 13.26. an.

```
DCL  A BINARY FIXED(31),
     P POINTER ;
DCL  1 B BASED(P),
       2 C BINARY FIXED(15),
       2 D CHARACTER (A REFER(C)) ;
A = 36 ;
LOCATE B FILE(...) ... ;
```

Abb. 13.26. Beispiel für eine basisbezogene Struktur mit einer Zeichenkettenvariablen unbestimmter Länge

Die LOCATE-Anweisung bewirkt in diesem Fall zweierlei:

1) Sie weist der Zeichenkettenvariablen D eine Länge von 36 zu, da A diesen Wert infolge der vorangegangenen Ergibtanweisung besitzt.

2) Sie setzt C auf den Wert von A.

Die REFER-Option kann unter einer der folgenden Voraussetzungen angewendet werden:

1) Sie erscheint nur bei Elementen auf der logischen Stufe 2.

2) Es liegt kein Auffüllen innerhalb der Struktur zwischen Strukturelementen vor.

Diese beiden Einschränkungen können vergessen werden, wenn die REFER-Option nur einmal benutzt wird und entweder auf die Länge einer Kette bezogen wird, die als

letztes Strukturelement in der Struktur vorhanden ist, oder auf die erste Dimension eines Bereiches, der das letzte Element der Struktur enthält. In der Abb. 13.27. finden wir ein diesbezügliches Beispiel.

```
DECLARE  1 STRUKTUR BASED(Z),
           2 M,
           2 N,
           2 Q(U REFER(M)) CHARACTER(V+2 REFER(N)),
           2 R BINARY FIXED(31) ;
DECLARE  (U,V) DECIMAL FIXED(4,2) ;
```

Abb. 13.27. Beispiele für die REFER-Option

Ein nachfolgendes LOCATE-Statement würde nun veranlassen, daß die Werte für M und N als ganze Zahlen bestimmt und als Dimensionsgrenze für Q sowie als Länge für die Bereichselemente benutzt werden. Wenn die REFER-Option zur Kenntlichmachung von Dimensionsgrenzen eingesetzt werden soll, so ist das nur für die obere Grenze möglich.

Nachdem einmal einer basisbezogenen Variablen Speicherplatz zugeordnet ist, so können selbstverständlich die Längen oder die Dimensionsgrenzen, die durch die REFER-Option gesetzt wurden, anschließend noch vermindert werden, sie dürfen jedoch nicht über den ursprünglichen Wert hinaus anwachsen. Die Änderung kann dadurch vorgenommen werden, daß M, N oder irgendeine Variable, die die Dimensionsgrenzen bzw. Längen enthält, wertmäßig variiert wird. Die ursprünglichen Werte müssen jedoch vor der Freigabe des zugeordneten Puffers bzw. Hauptspeicherplatzes wieder hergestellt werden, d.h. beispielsweise vor Ausführung der nächsten LOCATE-Anweisung.

Durch Niederschrift von V+2 in unserem Beispiel (Abb. 13.27.) wollten wir sagen, daß auch der Wert eines Ausdrucks einer die Länge bzw. die Dimensionsgrenze bestimmenden Variablen zugewiesen werden kann. Es bestehen freilich Einschränkungen: Der Ausdruck muß in eine ganze Zahl umgewandelt werden können und darf selbst keine Variablen enthalten, die in der basisbezogenen Struktur vorkommen.

13.17. Gebiete im Hauptspeicher

Mitunter ist es nützlich, wenn mehrere basisbezogene Strukturvariablen zur Verfügung stehen, die einander identisch sind. Durch das Statement

```
ALLOCATE bv [SET(zeiger)] ;
```

erreichen wir, daß Speicherplatz für die basisbezogene Strukturvariable bv zugeordnet wird. Wird die SET-Option weggelassen, so wird diejenige Zeigervariable mit einem Wert versorgt, die in der Erklärung der basisbezogenen Variablen aufgeführt ist. Betrachten wir zunächst das in der Abb. 13.28. dargestellte Beispiel.

Dieser kleine Programmabschnitt veranlaßt, daß drei identische Strukturen zur gleichen Zeit zur Verfügung stehen. Um sich auf die Strukturelemente innerhalb einer bestimmten Struktur beziehen zu können, müssen wir eine Zeigerkennzeichnung benutzen. Eine Ausnahme hiervon macht lediglich der Zeiger P, d.h. der Zeiger, mit dem die Struktur A erklärt ist: unterstellen wir ihn, so können wir auf die Kennzeichnung verzichten. Wie sieht nun eine Kennzeichnung über Zeiger aus? Hierzu ist ein Symbol, das Zeigerkennzeichnungssymbol, eingeführt worden, das zwei aufeinanderfolgende

```
DECLARE  (P,Q,R) POINTER,
         1 A BASED(P),
           2 B ··· ;
ALLOCATE A ;
ALLOCATE A SET(Q) ;
ALLOCATE A SET(R) ;
```

Abb. 13.28. Beispiel für die Zuordnung von Hauptspeicherplatz für basisbezogene Variable

Stellen belegt und aus dem Minuszeichen und dem Größer-Als-Zeichen besteht. Beide aneinandergereiht sehen dann wie ein Pfeil aus. Doch sehen wir uns Beispiele an:

```
Q -> B = ··· ;
XYZ = R->B + B ;
```

Beim letzten Statement werden die Werte addiert, die einmal zum Strukturelement B in der Struktur gehören, auf die die Zeigervariable R verweist, und zum anderen zum Strukturelement B in der Struktur, auf die die in die Erklärung von A einbezogene Zeigervariable P verweist. Erinnern wir uns, auch bei den READ-Anweisungen mit der SET-Option und den LOCATE-Anweisungen wird basisbezogenen Variablen Speicherplatz zugeordnet, in der gleichen Weise wie durch die ALLOCATE-Anweisung.

Nach der Zuordnung von Speicherplatz kann derselbe auch wieder zu anderweitiger Benutzung freigegeben werden, wozu das Statement

```
FREE [zeiger->] bv
     [[zeiger->] bv] [,···] ;
```

dient.

Beziehen wir uns auf das Beispiel von Abb. 13.28., so können wir beispielsweise den Speicherplatz für alle drei basisbezogenen Strukturen wieder durch das Statement

```
FREE A, Q->A, R->A ;
```

freigeben. Automatisch wird zugeordneter Speicherplatz durch READ-Anweisungen (mit der SET-Option), durch LOCATE- und durch CLOSE-Anweisungen freigegeben.

Wenn es wünschenswert ist, mehrere Generationen basisbezogener Variablen zusammen zu halten, so können wir das Attribut AREA benutzen, um ein benanntes Hauptspeichergebiet zu definieren. Die DCL-Vereinbarung für eine solche Gebietsvariable nimmt folgende Gestalt an:

```
DCL  gv AREA [(gebietsumfang)] ;
```

Unter gv verstehen wir hierbei eine Gebietsvariable. Gebietsvariable gelten als eine besondere Klasse innerhalb der Programmsteuerungsdaten; sie können weder mit anderen Datenarten verglichen noch ihre Werte ihnen zugewiesen werden. Der Umfang des Gebietes ist durch das Attribut gebietsumfang festzulegen. Fehlt dieses Attribut, so wird standardmäßig 1000 angenommen. Der maximale Wert für dieses Attribut ist 32767. Tatsächlich wird ein um 16 Bytes größeres Gebiet reserviert; die zusätzlichen 16 Bytes werden für Steuerzwecke benötigt.

Um anzugeben, daß einer basisbezogenen Variablen innerhalb eines Gebietes Speicherplatz zuzuordnen bzw. ihr zu entziehen ist, müssen wir die Option

```
IN(gv)
```

den ALLOCATE- bzw. FREE-Anweisungen hinzufügen. Die IN-Option im FREE-Statement wird dann und nur dann benötigt, wenn sie auch im zugehörigen vorangehenden ALLOCATE-Statement benutzt wurde. Ist innerhalb eines Gebietes nicht mehr genügend Platz vorhanden, um eine Hauptspeicheranforderung befriedigen zu können, so kommt es zum Auftreten der Bedingung AREA, die durch eine ON-Anweisung erfaßt werden kann und damit durch eine Eigenmaßnahme abgedeckt werden kann.

Ein Gebiet kann für alle Speicherklassen erklärt werden. Im besonderen kann eine Gebietsvariable selbst auch basisbezogen sein; beispielsweise wäre das durch

```
DCL   AB AREA BASED(Z),
      Z POINTER ;
```

der Fall. In einem solchen Fall muß der Variablen AB selbst erst Speicherplatz zugeordnet werden, bevor im durch sie repräsentierten Gebiet anderen Variablen Platz reserviert werden kann. Unter diesem Aspekt ist es wohl klar, daß die beiden Anweisungen

```
ALLOCATE AB ;
```

und

```
ALLOCATE A IN(AB) ;
```

logisch aufeinander folgen müssen. Die Freigabestatements lauten entsprechend

```
FREE A IN(AB) ;
```

und

```
FREE AB ;
```

Besondere Bedeutung besitzen mehrfache basisbezogene Strukturen in der sogenannten Listenverarbeitung, einer speziellen äußerst nützlichen und interessanten Programmierungstechnik. Nehmen wir einmal an, daß wir einen Zeitungsartikel analysieren wollen. Jeder Satz wird einer getrennten basisbezogenen Struktur zugewiesen. Der Redakteur beabsichtigt nun, Änderungen vorzunehmen. Er entfernt zu diesem Zweck einige Strukturen und fügt andere hinzu. Wenn nun jede Struktur einen Zeiger (Vorwärtszeiger) beinhaltet, der zur nächstfolgenden Struktur verweist, und einen zweiten Zeiger (Rückwärtszeiger) besitzt, der auf die vorhergehende Struktur verweist, so kann der Redakteur nach Beendigung seiner Korrekturen den Artikel leicht wieder zusammensetzen. Ein solches Vorgehen erfordert keine Datenübertragungen während des Korrekturprozesses. Die Listenverarbeitung ist, wie aus diesen Andeutungen hervorgeht, sicher eine recht nützliche Verarbeitungstechnik. Die Sprachelemente von PL/1, die sie unterstützen, gehören zu den einfachsten und am bequemsten zu handhabenden, die jemals für diese Vorgehensweise entworfen wurden.

13.18. Die gesteuerte Speicherklasse

Wir sind bisher drei verschiedenen Speicherklassen begegnet, nämlich folgenden:

Speicherklasse	Attribut
automatisch	AUTOMATIC
statisch	STATIC
basisbezogen	BASED

Es gibt noch eine vierte Speicherklasse, die als gesteuerte Speicherklasse bezeichnet wird. Variable, die zu dieser Klasse gehören sollen, sind durch das Attribut CONTROLLED zu kennzeichnen.

Die Zuordnung von Speicherplatz für derartige Variablen erfolgt ausschließlich durch Gebrauch der ALLOCATE-Anweisung, die Freigabe von Speicherplatz entsprechend durch die FREE-Anweisung. Eine einzige ALLOCATE- bzw. FREE-Anweisung kann sich sowohl auf basisbezogene, als auch auf der gesteuerten Speicherklasse zugehörende Variable beziehen. Von den Variablen, die zur gesteuerten Speicherklasse gehören, steht stets nur die zuletzt erzeugte Datengeneration zur Benutzung zur Verfügung, alle vorhergehenden Generationen werden zurückgestellt, d.h. gestapelt, und erst dann wieder zur Benutzung hervorgeholt, wenn die spätere Generation durch ein FREE-Statement eliminiert wurde. Veranschaulichen wir uns einen solchen Stapelprozeß durch das in Abb. 13.29. demonstrierte simple Beispiel mit nur zwei Generationen der gleichen zur gesteuerten Speicherklasse gehörenden Sturkturvariablen A.

```
DECLARE  1 A CONTROLLED,
           2 B ··· ;
ALLOCATE A ;
B = 35 ;              /* STRUKTUR A MIT STRUKTURELEMENT B GLEICH 35 */
ALLOCATE A ;
B = 65 ;              /* STRUKTUR A MIT STRUKTURELEMENT B GLEICH 65 */
FREE A ;              /* HERVORHOLEN DER STRUKTUR A MIT B GLEICH 35 */
FREE A ;              /* KEINE STRUKTUR A MEHR VERFUEGBAR */
```

Abb. 13.29. Beispiel für die Einrichtung eines Rückstellstapels

Die eingefügte Funktion ALLOCATION kann dazu benutzt werden, um festzustellen, ob für eine der gesteuerten Speicherklasse angehörenden Variablen gerade Speicherplatz zugeordnet ist. Beispielsweise ergibt sich bei der Anweisung

```
IF ALLOCATION(A)  THEN ··· ;
```

ein Betreten der THEN-Klausel, wenn A Speicherplatz zugeordnet ist. Der entstehende Funktionswert von ALLOCATION(A)wirkt nämlich so, als ob '1'B das Ergebnis des Ausdruckes in der IF-Anweisung ist. Bei nicht zugeordnetem Speicherplatz wird die THEN-Klausel übergangen bzw. bei Vorhandensein einer ELSE-Klausel wird diese betreten.

Im FREE-Statement für Variable, die der gesteuerten Speicherklasse zugehören, darf weder die IN-Option noch eine Zeigervariable erscheinen; bei der ALLOCATE-Anweisung muß entsprechend auf die IN-Option und die SET-Option verzichtet werden.

13.19. Das Einschließungsstatement zur Vorübersetzerzeit

Die Vorübersetzer-Statements von PL/1 sind Statements, die nicht in die Maschinensprache übersetzt, die vielmehr nur zur Übersetzerzeit ausgeführt werden. Derartige Statements gestatten es u.a. dem Programmierer, bestimmte Prozeduren und Datenvereinbarungen aus existierenden Bibliotheken in sein Programm aufzunehmen. Sie ermöglichen es weiterhin, Quellenprogramme mit einem Minimum an manuellen

Eingriffen zu modifizieren. Wenn eine Vorübersetzung verlangt wird, so liest der Kompilierer zunächst das codierte Programm und vollzieht alle Vorübersetzerstatements. Auf diese Weise erzeugt er ein reguläres Quellenprogramm, das dann anschließend normal übersetzt wird.

Vorübersetzerstatements unterscheiden sich von den normalen Statements dadurch, daß sie mit dem Prozentzeichen beginnen. Wir wollen in diesem Buch allein das Einschliessungsstatement betrachten. Es ist sicher das einzige Vorübersetzerstatement, das auch der Anfangsprogrammierer schon nutzbringend einsetzen kann. Dieses Statement besitzt die Form:

```
%[marke:] ... INCLUDE { bn(gn) } [ ,bn(gn) ] ... ;
                      { gn     } [ ,gn     ]
```

Hierbei bedeuten bn den Namen einer Bibliotheksdatei und gn den Namen eines in der Bibliotheksdatei enthaltenen Gliedes.

Wird während der Vorübersetzung ein %INCLUDE-Statement entdeckt, dann nimmt an dieser Stelle der Vorübersetzer den gesamten Inhalt von gn ins Quellenprogramm auf. Das eingeschlossene Bibliotheksglied kann selbst erneut Einschließungsstatements enthalten. Der ins Programm von gn aufgenommene Text wird gerade so betrachtet, als ob er unmittelbar vom Programmierer geschrieben und in sein Quellenprogramm eingefügt worden wäre.

Unter gn wird, wir deuteten es schon an, der Name eines Gliedes einer Bibliothek verstanden. Wenn im Betriebssystem OS bn niedergeschrieben wird, so wird dieser Name als Dateiname [1] angesehen; fehlt die Angabe von bn, so wird als Dateiname SYSLIB unterstellt [2]. Im Betriebssystem DOS wird immer die Quellenprogrammbibliothek SYSSLB benutzt, es sei denn, daß SYSSLB nicht zugeordnet ist. In diesem Fall tritt SYSRES an Stelle von SYSSLB. Ist im DOS ein Bibliotheksname bn aufgeführt, so darf er aus nur einem Buchstaben bestehen; fehlt die Angabe von bn, so wird P angenommen. Typische Einschließungsstatements könnten etwa wie folgt aussehen:

```
OS  | %INCLUDE  STRUKTUR, OSLIB(SIRUP) ;
DOS | %INCLUDE  STRUKUTR, R(SIRUP) ;
```

Am häufigsten werden Einschließungsstatements dazu verwendet, um in ein Programm standardisierte Strukturerklärungen für oft benutzte Dateien aufzunehmen. Dadurch, daß alle in einem Arbeitsgebiet tätigen Programmierer den gleichen Code benutzen, ist letzten Endes ja sichergestellt, daß alle Programme, die zu diesem Arbeitsgebiet gehören, auch die jüngsten und damit aktuellen Beschreibungen aufweisen.

13.20. Die Übergabe von Argumenten an PL/1-Hauptprogramme

Bei Benutzung des Betriebssystems OS, jedoch nicht bei Benutzung des Betriebssystems DOS, kann ein einziges Argument an ein PL/1-Programm übergeben werden. Dieses Argument ist in die PARM-Option des EXEC-Steuerstatements der Jobsteuersprache (JCL) aufzunehmen. Um Platz für die Aufnahme des Argumentes im Hauptprogramm zu schaffen, muß die PROCEDURE-Vereinbarung auf einen Parameter verweisen. Der Parameter selbst muß im Hauptprogramm als variabel lange Zeichenkettenvariable mit der Maximallänge 100 erklärt werden. Ein Hauptprogramm, dem ein Argument übergeben werden soll, muß also so gestaltet werden, wie es die Abb. 13.30. zeigt.

1) ddname in der Jobsteuersprache (JCL).

2) Für bn bzw. für SYSLIB muß in der Jobsteuersprache deshalb ein DD-Statement vorliegen.

```
MARIAN:  PROCEDURE(PARAM) OPTIONS(MAIN) ;
           DCL  PARAM CHARACTER(1ØØ) VARYING ;
           :
         END MARIAN ;
```

Abb. 13.30. Beispiel für die Präparierung eines Hauptprogrammes zur Übernahme eines im EXEC-Statement der JCL bereitgestellten Argumentes

Wenn die von der IBM unterstützten katalogisierten Prozeduren benutzt werden, so haben wir ins EXEC-Statement

PARM. GO='argument'

aufzunehmen. Wird jedoch der Lader benutzt, so müssen wir

PARM. GO='lader/argument'

vorsehen. Die vor dem Schrägstrich stehenden Argumente lader gelten dann für den Lader, das auf den Schrägstrich folgende für das Objektprogramm, d.h. für die mit OPTIONS(MAIN) gekennzeichnete Prozedur.

Aufgabengruppe 35

1) Gegeben ist die Vereinbarung

```
DECLARE  G DECIMAL FIXED(5,2) ;
```

Welche der nachfolgenden Vereinbarungen sind gültige Vereinbarungen?

```
a) DCL  H DECIMAL FIXED(7,2) DEFINED G ;
b) DCL  J DECIMAL FIXED(3,2) DEFINED G ;
c) DCL  K DECIMAL FIXED(5,Ø) DEFINED G ;
d) DCL  (L,M) DECIMAL FIXED(5,2) DEFINED G ;
e) DCL  N PICTURE'999V99' DEFINED G ;
f) DCL  P FIXED(5,2) DEFINED G ;
```

2) Gegeben ist die Vereinbarung

```
DCL  U CHARACTER(8) INITIAL('XX') ;
```

Welche der nachfolgenden Statements sind gültige Vereinbarungen?

```
a) DCL  A CHAR(8) INITIAL('XX') DEFINED U ;
b) DCL  B PIC'9,999V.99' DEFINED U ;
c) DCL  C CHAR(5) DEF(U) ;
d) DCL   1 D,  2 D2 CHAR(5) DEF(U),  2 D3 CHAR(3) ;
e) DCL   1 E DEFINED U,  2 E2 PIC'$99V.99',  2 E3 CHAR(1) ;
f) DCL   1 F DEF(U),  2 F2(4) CHAR(2) ;
g) DCL  G(3) CHAR(3) DEFINED U ;
h) DCL  H(2) CHAR(3) DEFINED U ;
```

3) Welche der unter a) bis d) aufgeführten Neudefinitionen sind gültig, wenn sie auf die Vereinbarung

```
DCL  R(4,5) CHAR(17) ;
```

bezogen werden?

```
a) DCL  S(4,5) CHAR(16) DEF R ;
b) DCL  T(4,3) CHARACTER(17) DEFINED(R) ;
```

c) `DCL U(4,2SUB+1SUB-3) CHARACTER(16) DEFINED R ;`
d) `DCL V(5,4) CHAR(14) DEFINED R(2SUB,1SUB) ;`

4) Es liegt folgende Strukturvereinbarung vor:

```
DCL  1 A,
       2 B DECIMAL FLOAT(14),
       2 C(3) DECIMAL FIXED(3),
       2 D BINARY FIXED(23),
       2 E,
         3 F(3) DECIMAL FIXED(3),
         3 G BINARY FIXED(23) ;
```

a) Welche Länge besitzt die Struktur insgesamt?
b) Wo befinden sich in der Struktur Füllbytes?

5) Bei der folgenden Struktur ist der Platz der Füllbytes und ihre Gesamtlänge zu ermitteln!

```
DCL  1 J,
       2 K DEC FLOAT(3),
       2 L PIC'999V.99',
       2 N POINTER,
       2 P DEC FLOAT(12),
       2 O,
         3 R CHAR(3),
         3 S POINTER,
       2 T CHAR(3),
       2 U PIC'999' DEFINED T ;
       2 V LABEL ;
```

6) Gegeben ist die folgende Vereinbarung:

```
DCL  A CHAR(4),
     B CHAR(7),
     C BIT(8),
     D PIC'-9V.99',
     E BIN FIXED(2Ø),
     F DEC FIXED(5,3),
     G DEC FLOAT(5) ;
```

Welche Zuweisung erfolgt jeweils durch die letzte Ergibtanweisung der folgenden Anweisungspaare?

```
a) C='Ø1'B;      B=C;
b) C='Ø1'B;      E=C;
c) C='Ø1'B;      F=C;
d) C='Ø1'B;      G=C;
e) C='Ø1'B;      D=C;
f) A='11Ø1';     B=A;
g) A='11Ø1';     C=A;
h) D=-37.28;     B=D;
i) D=-37.28;     A=D;
```

```
j) D=-37.28;     E=D;
k) D=-37.28;     G=D;
l) E=-8195;      C=E;
m) E=-8195;      G=E;
n) F=72.378;     D=F;
o) F=72.378;     E=F;
p) G=.ØØ7;       C=G;
q) G=.ØØ7;       D=G;
r) G=.ØØ7;       F=G;
```

7) Welche Zuweisung erfolgt durch die jeweils letzte Ergibtanweisung der nachfolgenden Paare?

```
a) A='11Ø1';     D=A;
b) A='11Ø1';     E=A;
c) B='ØØ2.ØØ3';  F=B;
```

```
d) E=-8195;      B=E;
e) F=72.378;     B=F;
f) G=.ØØ7;       B=G;
```

Zugrundegelegt ist die Vereinbarung von Aufgabe 6).

8) Unter Zugrundelegung der Vereinbarung von Aufgabe 6) ist anzugeben, welche Zuweisung durch die jeweils letzte der nachfolgenden Paare von Ergibtanweisungen veranlaßt wird!

a) B='8';	B=B+B;	d) A='2.39';	F=A+A;
b) B='8';	E=B+B;	e) G=.ØØ7;	E=G*1.ØE+3;
c) B='2.39';	D=A+A;	f) G=.ØØ7;	E=G**(-1);

9) Welcher Wert wird durch das nachfolgende Programm gedruckt?

```
GRI:  PROCEDURE OPTIONS(MAIN) ;
        DECLARE  (P,Q(1ØØ)) POINTER,
                 N BINARY FIXED(15),
                 D CHARACTER(1ØØ),
                 1 A BASED(P),
                   2 B BINARY FIXED(15),
                   2 C CHARACTER(N REFER(B)) ;
        D = (1Ø)'Ø123456789' ;
        DO  N=1 TO 1ØØ ;
            ALLOCATE A ;
            Q(N) = P ;
            C = D ;
        END ;
        DO  N=1 TO 1ØØ ;
            SUBSTR(D,1Ø1-N,1) = SUBSTR(C,N,1) ;
        END ;
        PUT PAGE LIST(D) ;
      END GRI ;
```

10) Wie ändert sich die Ausgabe von Aufgabe 9), wenn unmittelbar hinter das Statement

```
ALLOCATE A ;
```

die nachstehende Statementfolge eingefügt wird?

```
DECLARE  R PICTURE'999',
         S CHARACTER(3) DEFINED(R) ;
R = N ;
D = REPEAT(SUBSTR(S,3,1),1ØØ) ;
```

11) Wenn wir unmittelbar hinter die zweite DO-Anweisung des Programmes von Aufgabe 9) die Anweisung

```
P = Q(N) ;
```

einfügen, so hat das die Ausgabe eines anderen Wertes zur Folge. Wie sieht die Ausgabe aus?

12) Welche Beziehungen bestehen zwischen der Eingabe und der Ausgabe im folgenden Programm?

```
MARIAN:  PROCEDURE OPTIONS(MAIN) ;
         /* PROGRAMMIERT VON HORST ELMAR --- 31.12.1973 */
         DECLARE  (X,Y) DECIMAL FLOAT(16) ;
         DECLARE  NE  ENTRY (DECIMAL FLOAT(16)) RECURSIVE
                      RETURNS (DECIMAL FLOAT(16)) ;
         ON ENDFILE(SYSIN)  GOTO E ;
   NEU:  GET FILE(SYSIN) LIST(X) ;
         Y = NE(X) ;
         PUT SKIP FILE(SYSPRINT) LIST(X,Y) ;
         GOTO NEU ;
     E:  PUT FILE(SYSPRINT) LIST('PROGRAMM  ''MARIAN'' BEENDET')
           SKIP(6) ;
       END MARIAN ;
```

```
   NE:  PROCEDURE(F)  RETURNS (DECIMAL FLOAT(16)) ;
        /* PROGRAMMIERT VON HORST ELMAR --- 31.12.1973 */
        DECLARE  F DECIMAL FLOAT(16) ;
        IF  F<2  THEN RETURN(F) ;
                 ELSE RETURN(F*NE(F-2)) ;
      END NE ;
```

Als Eingabewerte sind nur natürliche Zahlen zugelassen. Verfolge den Programmablauf mit folgenden Eingabewerten:

a) 5 b) 6 c) 1 d) 2

14. Einführung in die FORTRAN-Sprache

14.1. FORTRAN

Das Wort FORTRAN ist eine Zusammenziehung aus den Wörtern FORmula TRANslation[1]). Zu einem FORTRAN-System gehören:

1) die im Jahre 1954 zum ersten Mal veröffentlichte Sprache, die sowohl Elemente der mathematischen Fachsprache als auch Wörter der englischen Sprache enthält,

2) ein in der Maschinensprache vorliegendes Programm, Kompilierer genannt; die Eingabe für dieses Programm stellt das in der FORTRAN-Sprache geschriebene Quellenprogramm dar, als Ausgabe entsteht ein äquivalentes Programm in der Maschinensprache.

Einige weitere, teilweise schon erwähnte Begriffe, sollen noch einmal festgehalten werden:

Unter dem Quellenprogramm oder Primärprogramm zur Lösung irgendeines Problemes versteht man ein in der FORTRAN-Sprache geschriebenes Programm.

Unter dem Objektprogramm oder Sekundärprogramm versteht man das Programm in der Maschinensprache, das vom FORTRAN-Kompilierer erzeugt wurde.

Um ein FORTRAN-Programm möglichst bequem schreiben zu können, werden für die praktische Codierarbeit meist verschiedene Arbeitshilfen zur Verfügung gestellt, wie Programmschemen, spezielle Lochkarten mit einem der FORTRAN-Sprache angepaßtem Aufdruck usw.

Durch die jahrelange Verwendung der FORTRAN-Sprache bedingt, haben sich in der Praxis verschiedene lässige Redensarten eingebürgert, an und für sich ein Zeichen dafür, in welch außerordentlichem Umfang sich die FORTRAN-Sprache durchgesetzt hat. So spricht man, um bei einem Beispiel zu bleiben, einfach von einem in FORTRAN geschriebenem Programm, statt daß man korrekt von einem in der FORTRAN-Sprache geschriebenem Programm spricht.

Alle für die FORTRAN-Sprache vorgenommenen Definitionen berühren das System IBM /370 in keiner Weise. Tatsächlich besteht eines der Hauptmerkmale der FORTRAN-Sprache darin, daß sie weitgehend unabhängig von irgendeinem speziellen Kompilierer ist. Man sagt deshalb, daß FORTRAN eine maschinenunabhängige Sprache ist und zur Klasse der problemorientierten Sprachen gehört.

Die FORTRAN-Sprache wurde über viele Jahre hinweg ständig weiterentwickelt; die heutzutage am meisten verwendete Version ist FORTRAN IV. Diese steht auch für das System IBM /370 zur Verfügung. Diese Version der Sprache wurde auch der Normung zugrundegelegt. Durch die USASI [2]) wurden zwei Sprachebenen genormt: FORTAN und BASIC FORTRAN. BASIC FORTRAN ist ein vollständiges Subset der vollen FORTRAN-Sprache, d.h. alle in BASIC FORTRAN geschriebenen Programme sind gleichzeitig auch FORTRAN-Programme.

Für das System IBM /370 stehen deshalb auch Kompilierer zur Verfügung, die beiden Sprachebenen gerecht werden [3]).

---

1) Formula Translation —— Formelübersetzung

2) USASI —— United States of America Standards Institute

3) Die für das System IBM/370 zur Verfügung stehenden beiden Sprachebenen weisen z. T. erhebliche Erweiterungen über FORTRAN bzw. BASIC FORTRAN hinaus auf.

Bevor wir weitergehen, halten wir es für angebracht, einige grundsätzliche Anmerkungen zu machen:

1) Ein vom FORTRAN-Kompilierer generiertes Maschinenprogramm kann niemals die höchstmögliche Leistungsfähigkeit besitzen, wenn man die Programmablaufzeit und den erforderlichen Speicherbedarf als Maßstab nimmt. Die Erfahrung hat jedoch gezeigt, daß das Objektprogramm fast ebenso leistungsfähig ist wie ein Programm, das in der einem System entsprechenden maschinenorientierten Programmiersprache von einem durchschnittlichen Programmierer geschrieben wurde. (Diese Erfahrung wurde nicht erst beim System IBM /370 gewonnen.)

2) Die Programmierung eines Programmes in der FORTRAN-Sprache erfordert weit weniger Zeit als die Programmierung des gleichen Programmes in der entsprechenden maschinenorientierten Sprache, nach gewissenhaften Beobachtungen etwa nur ein Drittel, z. T. noch weniger.

3) FORTRAN ist nicht für alle Probleme gleich gut geeignet. Am ehesten lohnt sich der Einsatz der FORTRAN-Sprache für ingenieurtechnische und wissenschaftliche Berechnungen. Zur Lösung von Problemen bei der Buchführung und von anderen kaufmännischen Problemen ist jedoch die Verwendung der FORTRAN-Sprache nur bedingt geeignet. Das gleiche gilt auch für die Datenerfassung bei statistischen Aufgaben.

14.2. Das Schreiben von FORTRAN-Programmen

FORTRAN-Quellenprogramme werden gewöhnlich auf besondere FORTRAN-Programmschemen geschrieben; ein solches wird in Abb. 14.1. gezeigt. Jedes FORTRAN-Statement ist in die Spalten 7 bis einschließlich 72 zu schreiben. Damit man sich auf ein FORTRAN-Statement beziehen kann, ist es möglich, ein Statement durch eine Nummer zu kennzeichnen. Statementnummern müssen fünfstellige ganze Zahlen ohne Vorzeichen sein, also im Wertebereich von 00001 bis 99999 liegen. Führende Nullen bleiben hierbei unbeachtet und werden durch Leerzeichen ersetzt; somit stellen also 00123 und 123 einander identische Statementnummern dar. Die Nummern dürfen den Statements in beliebiger Reihenfolge zugewiesen werden, jedoch dürfen keine zwei Statements durch die gleiche Nummer gekennzeichnet sein. Es ist im allgemeinen vorzuziehen, nur solche Statements zu numerieren, auf die man sich im Programm beziehen muß. Statementnummern sind in die Spalten 1 bis 5 zu schreiben.

In der Regel belegt jedes Statement nur eine Zeile des Programmschemas; ist jedoch ein Statement zu lang, dann kann es auf sogenannten Folgezeilen fortgesetzt werden. Es sind maximal 19 Folgezeilen möglich. Folgezeilen werden dadurch erkannt, daß in der Spalte 6 ein vom Leerzeichen verschiedenes Zeichen steht. Obwohl in Spalte 6 ein beliebiges Zeichen geschrieben werden kann, ist es zweckmäßig und sinnvoll, die Folgezeilen fortlaufend zu numerieren, d. h. durch Zeichen in aufsteigender Sortierfolge zu kennzeichnen.

Kommentare dürfen an beliebiger Stelle in ein Quellenprogramm eingefügt werden; sie werden durch den Buchstaben C in Spalte 1 erkannt. Kommentare werden ebenfalls gedruckt, wenn das Quellenprogramm aufgelistet wird; sie werden jedoch vom Kompilierer in keiner Weise sonst verarbeitet.

Vom Kompilierer werden ebenfalls die Spalten 73 bis 80 ignoriert. Man verwendet sie deshalb oft zur Niederschrift der Programmbezeichnung oder um die Zeilen des FORTRAN-Programmes fortlaufend zu numerieren. Jede Zeile des Programmschemas wird gewöhnlich auf eine Lochkarte gelocht, damit sie in den Computer eingegeben

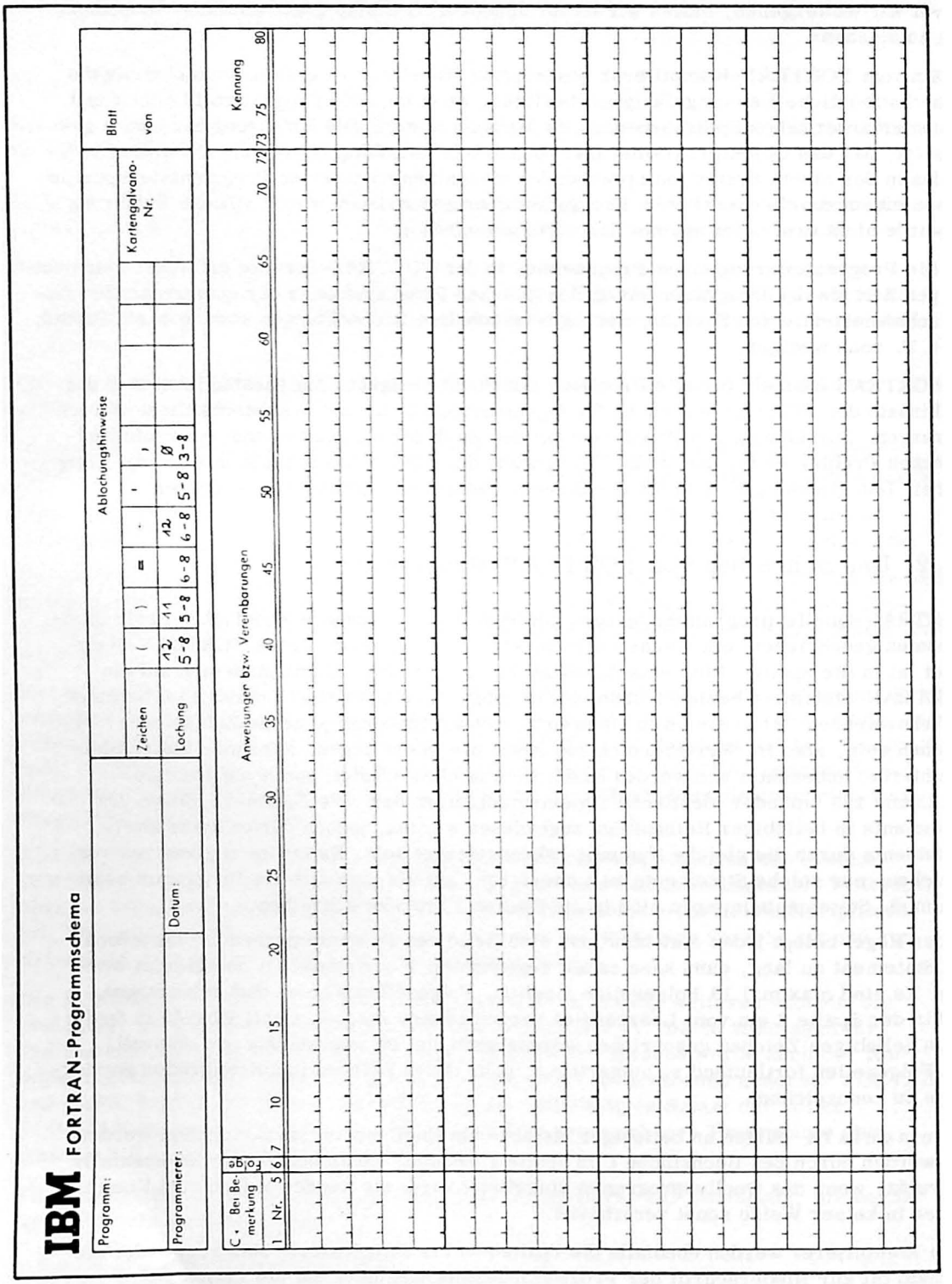


Abb. 14.1. FORTRAN-Programmschema

werden kann. Statt Lochkarten können aber auch andere Eingabemedien (Lochstreifen usw.) verwendet werden.

Wird ein Statement auf Folgezeilen fortgesetzt, so muß die Statementnummer, falls eine solche vorhanden ist, auf die erste Zeile geschrieben werden. Die Stellen 1 bis 5 von Folgezeilen müssen leer bleiben.

Es gibt fünf verschiedene Arten von FORTRAN-Statements:

a) Ergibtanweisungen
Aus diesen werden arithmetische und logische Maschineninstruktionen des Systems IBM /370 erzeugt.

b) Eingabe/Ausgabe-Anweisungen (EA-Anweisungen)

c) Steueranweisungen
Aus diesen werden Vergleichsinstruktionen bzw. Verzweigungsinstruktionen des Systems IBM /370 erzeugt.

d) Spezifikationsvereinbarungen
Aus diesen werden keine Maschineninstruktionen des Systems IBM /370 erzeugt. Sie dienen ausschließlich dazu, den Kompilierer zu informieren, wie und wo im Hauptspeicher die Daten zu speichern sind bzw. in welcher Form die ein- bzw. auszugebenden Daten vorliegen müssen bzw. aufgebaut werden.

e) Unterprogrammvereinbarungen
Diese werden im Kap. 16 behandelt.

Typische FORTRAN-Statements seien an einem kleinen Programm vorgestellt (Abb. 14.2.).

```
786 READ (1,234)  K,L,KJ,M,LUP
    IS = L + LUP + M
    IR = (KJ+2) / IS
    WRITE (3,234)  K,L,M,IS,IR
    GO TO 786
234 FORMAT (5I4)
    END
```

Abb. 14.2. Beispiel für FORTRAN-Statements

Die Statements FORMAT und END sind Spezifikationsvereinbarungen; die erste Vereinbarung legt fest, wie die Daten auf den Eingabekarten (oder auch auf anderen Eingabemedien) angeordnet sind, die Vereinbarung END dagegen verständigt den Kompilierer, daß im Quellenprogramm keine weiteren Statements vorhanden sind. Die Eingabe/Ausgabe-Anweisungen READ bzw. WRITE beziehen sich auf symbolische Namen von Daten und bewirken, daß Eingabe- bzw. Ausgabeoperationen ablaufen. Die Statements mit dem Ergibtzeichen (wird durch das mathematische Gleichheitszeichen symbolisiert) haben zur Folge, daß Werte berechnet und den Variablen zugewiesen werden, deren symbolische Namen IR und IS sind. Die Anweisung GO TO ist eine Steueranweisung; sie legt fest, daß als nächstes Statement das Statement mit der Nummer 786 auszuführen ist.

14.3. Konstanten und Variablen

Als ganzzahlige Konstante der FORTRAN-Sprache gilt jede beliebige Dezimalzahl, sofern sie keinen Dezimalpunkt enthält. Negativen Konstanten ist ein Minuszeichen voranzustellen. Eine ganzzahlige Konstante ohne Vorzeichen wird als positiv betrachtet. Beispiele für ganzzahlige Konstanten sind:

```
87
-9872
0
+1
89765490
```

Nachstehend folgen einige Beispiele für ungültige ganzzahlige Konstanten:

+87.	(der Dezimalpunkt ist nicht erlaubt)
-9,872	(das Komma ist nicht erlaubt)
0.893	(der Dezimalpunkt ist nicht erlaubt)

Im Objektprogramm wird jede ganzzahlige Konstante als Dualwert in einem Vollwort gespeichert. Der absolute Wert einer ganzzahligen Konstanten darf deshalb die Zahl

$$2^{31} - 1 = 2\ 147\ 483\ 647$$

nicht überschreiten.

Als reelle Konstante wird in der FORTRAN-Sprache jede Zahl angesehen, die einen Dezimalpunkt enthält. Beispiele hierfür sind:

```
87.
-9872.0
+1.
3.4578
0.03
-.00032456
```

Zusätzlich dürfen die reellen Konstanten noch einen Exponenten aufweisen. Dieser wird durch den Buchstaben E angezeigt, dem eine ganzzahlige Konstante mit einer Ziffernstelle oder mit zwei Ziffernstellen folgen kann. Dieser Exponent bedeutet, daß die vor dem E stehende Konstante mit der Potenz multipliziert wird, deren Basis 10 und deren Exponent gleich der Zahl ist, die dem Buchstaben E folgt. Beispiele reeller Konstanten, die in dieser Form geschrieben sind, folgen nachstehend:

-.0123E+3	($\hat{=}$ $-0,012 \cdot 10^{3}$)
9.3E07	($\hat{=}$ $9,3 \cdot 10^{7}$)
6.E36	($\hat{=}$ $6,0 \cdot 10^{36}$)
0.0E+0	($\hat{=}$ $0,0$)
-2.7E-3	($\hat{=}$ $-2,7 \cdot 10^{-3}$)
-0.45E-17	($\hat{=}$ $0,45 \cdot 10^{-17}$)
-17.2E1	($\hat{=}$ $-17,2 \cdot 10^{1}$)

Reelle Konstanten sind auf maximal sieben gültige Ziffern beschränkt. Im Objektprogramm wird jede reelle Konstante als Gleitpunktzahl in einem Vollwort gespeichert. Reelle Konstanten dürfen deshalb, absolut genommen, nur in dem Wertebereich liegen, der im System IBM /370 durch die zulässigen Exponenten von Gleitpunktzahlen festgelegt ist, d.h. kleiner als ungefähr 10^{75}.

Eine doppelt genaue Konstante [1] unterscheidet sich von einer reellen Konstanten dadurch, daß sie im Objektprogramm ein Doppelwort belegt. Doppelt genaue Konstanten werden entweder als Zahl mit einem Dezimalpunkt geschrieben, die 8 bis 16 gültige Ziffern aufweist oder als reelle Konstante, bei der der Exponent an Stelle des Buchstabens E durch den Buchstaben D angezeigt wird. Beispiele hierfür sind:

```
1234567.890123456
13579135000.
0.13579123D+11
9.3D7
0.0D0
-2.D-07
-.012D-3
```

Nachstehend folgen einige Zahlen, die keine gültigen doppelt genauen Konstanten darstellen.

```
123,456.789              (es ist ein Komma vorhanden)
12345678901.2345678      (mehr als 16 gültige Ziffern)
9.27D+4.                 (kein ganzzahliger Exponent)
```

Jede Zahl, die in einem FORTRAN-Statement explizit aufgeführt ist, gilt als Konstante; eine Zahl, auf die man sich mittels eines Namens beziehen muß, gilt als Wert einer Variablen. Somit ist in der Ergibtanweisung

```
IR = (KJ + 2)/IS
```

2 eine Konstante, während IR, KJ und IS Variable sind.

Die Namen, die man Variablen zuweist, können ein bis sechs Zeilen lang sein; das erste Zeichen muß dabei ein alphabetisches Zeichen sein, während die restlichen Zeichen alphabetisch oder numerisch sein können. In der FORTRAN-Sprache gilt das Währungszeichen $ als alphabetisches Zeichen, dadurch umfaßt das Alphabet insgesamt 27 Buchstaben. Die Namen können also aus einem Zeichenvorrat von 27 alphabetischen Zeichen und 10 numerischen Zeichen gebildet werden. Die Verwendung von Sonderzeichen, einschließlich des Leerzeichens, ist für Namen nicht erlaubt.

Wenn nicht anders festgelegt, so kennzeichnet ein mit den Buchstaben I, J, K, L, M oder N beginnender Name eine Variable, die ausschließlich nur ganze Zahlen als Werte annehmen kann. Für eine solche Variable werden im Objektprogramm 4 Bytes reserviert. Ist das erste Zeichen eines Namens irgendein anderer Buchstabe (A bis H, O bis Z, $), so repräsentiert die betreffende Variable eine reelle Größe, auch ihr werden im Objektprogramm 4 Bytes zugeordnet. Beispiele für ganzzahlige Variablen sind:

```
M
IBM37Ø
KUH7X
IFA
```

Beispiele für reelle Variablen sind:

```
BK35Ø
X
$78EF9
ANNE
```

1) Besser: „erhöhtgenaue" Konstante.

Im folgenden werden noch einige Zeichenketten aufgeführt, die keine gültigen Namen darstellen:

IBM 18	(in Namen ist kein Leerzeichen erlaubt)
IBMA1443	(mehr als 6 Zeichen im Namen)
6APP	(das erste Zeichen ist nicht alphabetisch)
R9.7	(der Name enthält ein Sonderzeichen)
F(X)	(der Name enthält Sonderzeichen)

Die Festlegung, daß der erste Buchstabe den Typ der Variablen bestimmt, kann durch eine explizite Typvereinbarung zunichte gemacht werden. Eine solche Typvereinbarung besitzt folgende allgemeine Formen:

INTEGER $v_1,v_2,\cdots,v_n$
REAL $v_1,v_2,\cdots,v_n$
DOUBLE PRECISION $v_1,v_2,\cdots,v_n$

Jede der in einer solchen Vereinbarung aufgeführten Variablen nimmt dann den angegebenen Typ ohne Rücksicht auf den ersten Buchstaben an. Es folgen einige Beispiele für Typvereinbarungen:

```
INTEGER  I87Y,ER,MARI
REAL  M91,ANNE
DOUBLE PRECISION  Y8,POI,K4
```

Durch diese Vereinbarungen wird erreicht, daß I87Y und ER als ganzzahlige Variablen eingeführt werden, M91 als reelle Variable und Y8, POI sowie K4 als doppelt genaue Variablen. Auf die Typvereinbarung von I87Y hätte man auch verzichten können, da alle Variablen, deren Name mit I beginnt, sowieso als ganzzahlige Variablen angesehen werden. Das gilt auch für den Namen MARI und bei den reellen Variablen für ANNE.

Doppelt genauen Variablen und Konstanten werden im Objektprogramm Doppelwörter zugeordnet, die zur Aufnahme von Gleitpunktzahlen dienen.

In der vollen Sprache ist es zusätzlich auch erlaubt, explizite Typvereinbarungen vorzunehmen, die es ermöglichen, den Umfang des Speicherplatzes anzugeben, der den betreffenden Variablen zugeordnet werden soll. Folgende Vereinbarungen sind möglich:

INTEGER*2 $v_1,v_2,\cdots,v_n$
INTEGER*4 $v_1,v_2,\cdots,v_n$
REAL*4 $v_1,v_2,\cdots,v_n$
REAL*8 $v_1,v_2,\cdots,v_n$

Schreibt man INTEGER*4 bzw. REAL*4, so sind diese Schreibweisen gleichwertig den Schreibweisen INTEGER bzw. REAL. Die Schreibweise REAL*8 ist gleichwertig der Schreibweise DOUBLE PRECISION. Die Niederschrift von INTEGER*2 hat zur Folge, daß den in dieser Vereinbarung genannten Variablen vom Kompilierer ein Halbwort zugeordnet wird. Die diesen Variablen zugewiesenen Werte werden folgerichtig als Dualzahlen gespeichert. Ihr Wertevorrat ist deshalb auf den Wertebereich von -32768 bis 32767 beschränkt.

In der vollen Sprache ist es ebenfalls erlaubt, für alle Variablen einer Liste gemeinsam den Speicherbedarf festzulegen (angegeben durch die Länge in Bytes), jedoch einzelne Variablen der Liste von dieser allgemeinen Länge auszuschließen, indem für sie eine spezielle, nur für sie gültige Länge angegeben wird. Beispielsweise werden durch die Vereinbarung

```
REAL*8    PR, PS, KL*4, KK
```

den Variablen PR, PS, und KK Doppelwörter zugeordnet, der Variablen KL dagegen nur ein einfaches Wort. In ähnlicher Weise wird durch

```
INTEGER   MK*2, RT
```

der Variablen RT ein volles Wort und der Variablen MK ein Halbwort zugeordnet. Beide werden jedoch als ganzzahlige Variablen eingeführt. Man erinnere sich, daß für MK der Wertebereich beschränkt auf 32767 ist, d.h. MK darf keine Werte annehmen, die absolut größer sind als diese Zahl.

14.4. Arithmetische Ausdrücke

Ein arithmetischer Ausdruck besteht aus einer beliebigen Folge von Konstanten, Variablen und Funktionen, die durch Operatoren (wie +, -, *usw.) bzw. Klammern so voneinander getrennt sind, daß ein gültiger mathematischer Ausdruck vorliegt. Fünf verschiedene Operatoren sind erlaubt:

Operator	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
**	Potenzierung

Zwar erfordert das Symbol ** zwei aufeinanderfolgende Spalten auf dem Programmschema, es wird aber als ein einziger Operator angesehen. Beispiele für Ausdrücke sind:

```
7 + K*4
B**2 - 4.*A*C
(1.+R/T)**(T*P)/R
3.
X/4.Ø
ABC + DE7*UE7
```

In einen Ausdruck können zur Verbesserung der Lesbarkeit beliebig viele Leerzeichen eingefügt werden, jedoch nicht zwischen die Zeichen eines Namens. Bei der Bildung der Ausdrücke sind folgende Regeln zu beachten:

1) Zwei aufeinanderfolgende Operatoren müssen durch eine Klammer voneinander getrennt werden. Der Ausdruck A/-B ist somit nicht korrekt, A/(-B) ist hingegen erlaubt.

2) Wird eine ganzzahlige Größe (Konstante oder Variable) zu einer ganzzahligen Potenz erhoben (Beispiel I**J), so ergibt sich ein ganzzahliges Resultat. In allen anderen Fällen ergibt sich ein reeller Potenzwert.

3) Enthält ein Ausdruck irgendeine reelle Größe, so ist der Wert des Ausdruckes ebenfalls reell. Enthält ein Ausdruck Größen ungleicher Länge (d.h. ihr durch die Länge in Bytes angegebener Speicherplatzbedarf ist unterschiedlich), so besitzt der Wert des Ausdruckes die größte Länge, die eine im Ausdruck vorkommende Größe besitzt. Man betrachte dazu folgendes Beispiel:

```
REAL   U
DOUBLE PRECISION   Z, Y
INTEGER   K
INTEGER*2   I
```

Der Ausdruck Z*Y ergibt einen Wert der Länge 8 Bytes, d.h. ein Doppelwort, und ist infolgedessen doppelt genau reell. Der Ausdruck U/K ist reell und besitzt eine Länge von 4 Bytes (ein Wort), K+I ergibt einen ganzzahligen Wert von 4 Bytes (ein Wort) Länge, desgleichen auch K**2.

4) Fehlen Klammern, die die Reihenfolge der Ausführung der arithmetischen Operationen eines Ausdruckes festlegen, so werden die arithmetischen Operationen in folgender Reihenfolge ausgeführt:

   a) Berechnung von Funktionen,
   b) Potenzierung **,
   c) Multiplikation * und Division / ,
   d) Addition + und Subtraktion -.

Bei arithmetischen Operationen gleicher Rangordnung werden die Operationen von links nach rechts ausgeführt, mit Ausnahme von Potenzierungen, die von rechts nach links fortlaufend ausgeführt werden. An einem Beispiel soll durch Setzen von Klammern gezeigt werden, in welcher Reihenfolge die Operationen ausgeführt werden:

```
A*B/C**D**E + 7.0*F                bedeutet also
((A*B)/(C**(D**E))) + (7.0*F)
```

Tatsächlich fügt der Kompilierer Klammern ein, bevor er mit der Übersetzung fortfährt. Wenn man über die Reihenfolge der durchzuführenden arithmetischen Operationen bei verwickelt aufgebauten Ausdrücken in Zweifel gerät, so sollte man zur Sicherheit lieber Klammernpaare einfügen, auch wenn sie nicht benötigt werden.

5) Wird eine ganzzahlige Größe durch eine andere ganzzahlige Größe dividiert, so wird nur der ganzzahlige Anteil des Quotienten festgehalten. Es wird keine Rundung vorgenommen. Wenn also I gleich 3 und J gleich 8 ist, dann ergibt J/I den Wert 2, da $8:3=2,\overline{6}\ldots$ ist und die Stellen nach dem Komma vernachlässigt werden. Berechnet man (-J)/I, so ergibt sich in gleicher Weise der Wert -2. Nehmen wir für ein weiteres Beispiel an, daß I=3, J=8 und A=4. 0 ist und daß wie zuvor keine expliziten Typvereinbarungen für die Variablen vorliegen, so ergibt der Ausdruck J/I*A den Wert 8.0, denn es wird zunächst J/I berechnet (Wert 2) und anschließend erst wird die Multiplikation mit A ausgeführt, die auf das Resultat 2*4.0=8.0 führt. Schreiben wir dagegen A*J/I, so wird zunächst A*J zu 4.0*8=32.0 berechnet und anschließend erst wird die Division 32.0/I zu 32.0/3=10.66667 ausgeführt. Wir erkennen daraus, daß für die Erhaltung eines bestimmten Resultates mitunter auch die Reihenfolge der Codierung des Ausdruckes von entscheidender Bedeutung ist.

6) Potenzen mit ganzzahligen Exponenten werden als wiederholte Multiplikationen ausgeführt (z.B. I**J oder A**J). Ist der Exponent dagegen reell, so erfolgt die Berechnung des Potenzwertes über die Logarithmen (z.B. I**A oder B**A).

7) Im allgemeinen werden Objektprogramme optimaler aufgebaut sein (d.h. laufen schneller ab und erfordern weniger Speicherplatz), wenn in einem Ausdruck reelle und ganzzahlige Größen nicht miteinander vermischt auftreten, eine Ausnahme bilden hierbei selbstverständlich Potenzierungen mit ganzzahligen Exponenten. Soll z.B. U+J berechnet werden, so muß nämlich der Kompilierer in das Objektprogramm Instruktionen aufnehmen, die vor der Durchführung der eigentlichen arithmetischen Operation zunächst einmal den Wert von J in eine Gleitpunktzahl umwandeln. Wenn es immer nur möglich ist, sollte man deshalb derartige Umwandlungen vermeiden. Daraus folgt sofort, daß X+4.0 dem Ausdruck X+4 vorzuziehen ist, ebenso ist X*3 dem Ausdruck X*3.0 unterlegen.

8) Es ist Vorsorge getroffen, daß zur Berechnung bestimmter Funktionen Unterprogramme in jedes FORTRAN-System eingeschlossen sind. Alle diese Funktionen sind im Anhang F aufgeführt, einige sollen aber hier schon erwähnt werden:

Funktion	Bedeutung
EXP(x)	exp x bzw. e^x
SQRT(x)	$\sqrt{x}$
ALOG(x)	ln x
ALOG10(x)	lg x
DLOG(x)	ln x

Zwischen ALOG und DLOG besteht folgender Unterschied: Für DLOG ist ein doppelt genaues Argument erforderlich, der Funktionswert ist ebenfalls doppelt genau, ALOG hingegen erfordert ein einfach genaues Argument (also eine reelle Größe) und ergibt ebenso einen einfach genauen Funktionswert.

Die Funktionen werden in zwei verschiedene Arten unterteilt, in die sogenannten offenen (in Linie liegenden) und in die sogenannten geschlossenen (außerhalb liegenden) [1]. Bei der ersten Art werden die zur Berechnung des Funktionswertes erforderlichen Maschineninstruktionen durch den Kompilierer immer an den Stellen ins Objektprogramm eingefügt, an denen die Funktion niedergeschrieben wird. Funktionen der zweiten Art werden grundsätzlich nur ein einziges Mal ins Objektprogramm aufgenommen; immer wenn der Funktionswert berechnet werden soll, verzweigt das Objektprogramm zu der Routine, die den Funktionswert berechnet; nach ihrer Ausführung erfolgt ein Rücksprung zum Hauptprogramm. Für den Programmierer besteht hinsichtlich der beiden Arten von Funktionen kein Unterschied beim Niederschreiben seines Programmes.

Die Instruktionsmenge, die zur Berechnung der verschiedenen Funktionen erforderlich ist, ist ein Teil der Systemresidenz-Bibliothek. Der Benutzer kann aus dieser Bibliothek Funktionen entfernen, aber auch neue hinzufügen.

Die Argumente der Funktionen müssen stets in Klammern gesetzt werden. Argumente von Funktionen dürfen Ausdrücke sein, die selbst wieder Funktionen enthalten; es ist hierbei selbstverständlich möglich, daß das Argument einer Funktion selbst wieder auf die gleiche Funktion Bezug nimmt. Es ist also z.B. die Bildung von Ausdrücken wie des nachstehenden möglich:

```
SQRT(ALOG(1.+SQRT(X**2+Y**2)))
```

1) Im Englischen: In-Line-Functions bzw. Out-of-Line-Functions.

14.5. Ergibtanweisungen

Eine arithmetische Ergibtanweisung besitzt die Form

a=b

Hierbei stellt a eine beliebige Variable und b einen Ausdruck dar. Aus einer solchen Ergibtanweisung erzeugt der Kompilierer eine Instruktionsmenge, die zunächst den Wert des Ausdruckes auf der rechten Seite des Ergibtzeichens berechnet und anschliessend diesen Wert auf den Speicherplatz überträgt, der der Variablen a zugeordnet ist, auf diese Weise den früheren Wert von a ersetzend. Somit hat das Ergibtzeichen (geschrieben in der Form eines mathematischen Gleichheitszeichens) eine Bedeutung im Sinne von „wird ersetzt durch". Dieses Zeichen drückt also unter keinen Umständen eine Gleichheit zweier Größen aus. Es folgen als Beispiele einige gültige Ergibtanweisungen:

Anweisung	Bedeutung
A=2.0	Speicherung von 2.0 als Wert von A
B=C	Speicherung des augenblicklichen Wertes von C nach B
D=2.*C	Speicherung des doppelten Wertes von C nach D
J=J+1	Addition von 1 auf den augenblicklichen Wert von J und Speicherung der Summe als neuen Wert nach J

Wenn der Wert des Ausdruckes auf der rechten Seite der Ergibtanweisung berechnet ist, wird er, wenn notwendig, in den Modus (reell oder ganzzahlig mit 2, 4 oder 8 Bytes Länge) umgewandelt, der die Variable auf der linken Seite des Ergibtzeichens auszeichnet. Erst danach wird er als neuer Wert auf den Speicherplatz übertragen, der der Variablen auf der linken Seite des Ergibtzeichens zugeordnet ist. Verfolgen wir mehrere Beispiele:

Es ist nach Voraussetzung A=2.0 , B=3.0 , J=9 , K=-2 und P=4.8.

a) C=B/A	1.5 wird als Wert unter C gespeichert.
b) I =B/A	1 wird als Wert unter I gespeichert
c) D=J /K	-4.0 wird als Wert unter D gespeichert
d) L=J /K	-4 wird als Wert unter L gespeichert
e) M=P/B	1 wird als Wert unter M gespeichert
f) R=P/B	1.6 wird als Wert unter R gespeichert
g) S=J/K/P	0.8333333 wird als Wert unter S gespeichert

Es ist hierbei angenommen worden, daß für die Variablen keine explizite Typvereinbarung erfolgt ist.

Es ergeben sich stets dann schnellere (dynamisch kürzere) Objektprogramme, wenn unnötige Umwandlungen zwischen reellen und ganzzahligen Größen in einem Programm vermieden werden.

Aufgabengruppe 36

1) Untersuche, ob die folgenden Konstanten gültige ganzzahlige oder reelle oder erhöht genaue Konstanten sind!

   a) -35 b) 02468 c) 7,329 d) 3.14159 e) 1234.56789012 f) 3.E83
   g) 431.2D32 h) .00000000000138E-12 i) 0 j) 9,368.894 k) 1.D+45 l) 3.D83

2) Untersuche, ob die folgenden Variablennamen für ganzzahlige oder reelle Variable gültig sind

a) DORA b) ZEIT c) N d) BK350 e) MACHT f) RETTUNG g) C/O
h) ARB BE i) I.30 j) $256 k) 2RATE l) G(Y)

3) Schreibe einen Ausdruck zur Berechnung von

a) ax^2+bx+c b) $a(b+c)$ c) $\frac{a}{b}+c$ d) $\frac{a}{b}.c$ e) $\sqrt{b^2-4ac}$
f) $\frac{(i^3+5)^2}{7-n}$

4) Es soll der Wert für die nachstehenden Ausdrücke angegeben werden. Vorausgesetzt ist, daß die Variablen folgende Werte angenommen haben: A = -2.0, B = 0.4, C = 7., D = 2., J = 2, K = -3

a) PA=J**K-K**J b) PB=A**J+A**K c) PC=A*C-EXP(2.5*B)
d) DP=SQRT(B*C+A) e) PE=A/B*C f) IF=A*C-EXP(2.5*B)
g) IG=J**K h) IH=D**K*C/B i) PI=A**J**K
j) PJ=(1.+B/D)**(D*J)/B k) PK=7+K*4 l) IL=K/J

5) Jede der folgenden Anweisungen ist falsch, ermittle die Fehler!

a) D=A**-K+J b) E+L=M*N/B c) F(X)=G**2+7 d) G=SQRT(-3.B+9)
e) G=SQRT(-3.0B**2) f) X+17.=Y g) 8PQ=7PQ8=
h) AMT=(((1-R)/2.+7)**0.26 i) SR=(PQ*2-PQ-PQ)**(-3)
j) EG=7.2E-4+8/96G

6) Der Flächeninhalt F eines Dreieckes mit den Seiten a, b, und c ist durch die Formel

$$F = \sqrt{s(s-a)\,(s-b)\,(s-c)}$$

gegeben. Hierbei bedeutet

$$s = \frac{a+b+c}{2}$$

Schreibe die Anweisungen, die erforderlich sind, um den Flächeninhalt eines Dreieckes zu berechnen. Die Seiten seien mit SDA, SDB und SDC bezeichnet.

7) Schreibe die erforderlichen Anweisungen zur Berechnung der Werte der Funktion cosh für das Argument $A^2 + B^2$.

14.6. Steueranweisungen

In welcher Reihenfolge das Objektprogramm vom Kompilierer zusammengestellt wird, hängt einzig und allein von der Reihenfolge der Karten ab, auf denen das Quellenprogramm abgelocht ist. Bei allen Objektprogrammen beginnt der Ablauf des Programmes mit der Ausführung der Instruktionen, die aus der ersten Anweisung des Quellenprogrammes erzeugt wurden. Danach geht die Programmablaufsteuerung von einer Anweisung zur nächsten über, es sei denn, daß eine Steueranweisung diese Reihenfolge ver-

ändert. In diesem Abschnitt wollen wir nunmehr die wichtigsten Steueranweisungen kennenlernen.

a) GO TO

Die Anweisung GO TO hat eine unbedingte (absolute) Verzweigung (Sprung) zur Folge. Sie besitzt die Form

GO TO n

Hierbei bedeutet n eine Statementnummer. Zum Beispiel bewirkt das Statement

```
GO TO 1786
```

daß das Statement, das durch die Statementnummer 1786 gekennzeichnet ist, als nächstes Statement ausgeführt wird.

Ist die Programmablaufsteuerung erst einmal zur neuen Stelle übertragen worden, werden von dieser aus die Statements wieder sequentiell ausgeführt, natürlich nur solange, solange keine neue Steueranweisung auftritt, die die Programmablaufsteuerung erneut verändert.

b) IF

Die Anweisung

IF (a) n_1,n_2,n_3

veranlaßt zunächst die Berechnung des in Klammern stehenden Ausdrucks a. Ist $a<0$, so erfolgt eine Verzweigung des Programmes zum Statement mit der Statementnummer n_1, ist $a=0$, so erfolgt die Verzweigung zum Statement mit der Statementnummer n_2, und ist schließlich $a>0$, so erfolgt die Verzweigung zum Statement, das durch die Nummer n_3 gekennzeichnet ist. Die IF-Anweisung ist also eine bedingte Verzweigungsanweisung. Analysiert man einmal eine Reihe von Programmen, so wird man feststellen, daß die IF-Anweisung eine der am häufigsten gebrauchten Anweisungen ist.

Wir wollen noch ein Beispiel betrachten:

```
IF (X-5.0) 3,6,3
```

Durch diese Anweisung wird erreicht, daß die Programmablaufsteuerung zum Statement mit der Nummer 6 verzweigt, wenn X-5.0=0 ist, d.h. wenn X=5.0 ist; in allen anderen Fällen erfolgt die Verzweigung zum Statement 3.

c) PAUSE

Das PAUSE-Statement kann in einer der nachfolgend aufgeführten drei Formen codiert werden:

PAUSE
PAUSE n
PAUSE 'nachricht'

Bei allen drei Formen wird vom Objektprogramm eine Nachricht an die Maschinenbedienung übermittelt. Die Programmausführung wird solange eingestellt, bis der Operateur eine Antwort eingegeben hat. Jede Antwort wird akzeptiert. Nach Annahme der Antwort setzt der Programmablauf mit der Ausführung der nächsten auf PAUSE folgenden Anweisung wieder ein. Die der Maschinenbedienung übermittelte Nachricht schließt bei der 3. Form den Text von nachricht (jedoch ohne die begrenzenden Ausführungszeichen) ein. In den beiden anderen Formen wird der Wert von n als vorzeichenfreie fünfstellige ganze Zahl ausgegeben; in der ersten Form wird systemintern n mit dem Wert 0 ergänzt.

Die PAUSE-Anweisung sollte in ein Programm dann und nur dann eingefügt werden, wenn es notwendig ist, das Programm anzuhalten, um manuelle Eingriffe an der Maschine vorzunehmen.

d) STOP
Jedes FORTRAN-Programm sollte mit dem STOP-Statement abgeschlossen werden. Diese Anweisung kann in zwei Formen benutzt werden:

STOP
STOP n

Wird n angegeben, so wird n ausgeschrieben. Unter n ist eine ganze Zahl ohne Vorzeichen mit nicht mehr als fünf Stellen zu verstehen. In jedem Fall wird beim Erreichen dieser Anweisung die Ausführung des Objektprogrammes abgebrochen und die Ablaufsteuerung zurück an das Betriebssystem gegeben, das dann mit der Vorbereitung für den Ablauf des nächsten Programmes beginnt. Wird n nicht angegeben, so wird nichts ausgeschrieben.

Die STOP-Anweisung ist die normale Methode, um die Ausführung eines in der FORTRAN-Sprache geschriebenen Programmes zu beenden.

e) END
Das Statement END muß das physisch letzte Statement jedes Quellenprogrammes sein. Durch dieses Statement wird dem Kompilierer signalisiert, daß das FORTRAN-Quellenprogramm keine weiteren Statements aufweist. Aus dem END-Statement werden keine Maschineninstruktionen generiert; es stellt also eine Vereinbarung dar. Die END-Vereinbarung darf nicht durch eine Statementnummer markiert sein.

Aufgabengruppe 37

1) Welcher Wert wird für ASIN im folgenden Programm gespeichert?

```
      SUM = Ø.Ø
      T = 1.Ø
    7 F = 1.Ø/(T**2+T
      SUM = SUM + F
      IF (F-1.ØE-6)  3,3,5
    5 T 3 T + 1.Ø
      GO TO 7
    3 ASIN = SUM
      STOP
      END
```

2) Welche Beziehung besteht zwischen den Variablen X und Y im folgenden Programm?

```
      IF (X-4.)  1,1,2
    1 IF (X)  3,4,4
    3 Y = X - 9.
      GO TO 789
    4 Y = 4.*X - 9.
      GO TO 789
    2 Y = 3.*X - 5.
  789 Y = Y + 1
      STOP
      END
```

3) Welche Antwort ist auf die Fragestellung der Aufgabe 2) zu geben, wenn die vor der Anweisung mit der Marke 2 stehende Anweisung

```
GO TO 789
```

entfernt wird?

4) Schreibe ein Programm zur Berechnung der Werte der Funktion

$$y = \begin{cases} 3x^2 + 7 & \text{für } x > 8 \\ 2x & \text{für } 1 \leqq x \leqq 8 \\ 2x^2 - 3 & \text{für } x < 1 \end{cases}$$

5) Schreibe ein Programm, das Y auf den gebrochenen Anteil von X setzt. Beispiele:
a) Wenn X = 1,78 ist, so soll Y auf 0,78 gesetzt werden.
b) Wenn X = 56.02 ist, so soll Y auf 0.02 gesetzt werden.

6) Gegeben sind die Werte für die fünf Variablen A1, A2, A3, A4 und A5. Schreibe ein Programm, mit dessen Hilfe der größte dieser Werte B1 zugewiesen wird, der nächstgrößere B2 usw., der kleinste demnach B5. Einige der Werte der Variablen A1, A2, A3, A4 bzw. A5 können einander gleich sein.

7) Gegeben sind X und Y. Schreibe ein Programm, mit dessen Hilfe Werte von RHO und von THETA berechnet werden können. RHO ist gleich der Quadratwurzel aus X^2 plus Y^2, THETA gleich dem Arcustangens von Y/X.

8) Schreibe ein Programm zur Bestimmung der Lösungen der quadratischen Gleichung $ax^2 + b\,x + c = 0$. Die beiden Lösungen sind unter X1 bzw. X2 zu speichern. Folgende Voraussetzungen sollen dem Programm zugrundeliegen:

a) Die Werte der Koeffizienten stehen bereits unter A, B und C zur Verfügung.
b) Die Koeffizienten können nur solche Werte annehmen, daß sich ausschließlich reelle Lösungen ergeben.

9) Schreibe ein Programm zur Berechnung der Lösungen des Gleichungssystems

I. $a\,x + by = c$
II. $d\,x + ey = f$

Besitzt dieses System keine Lösungen, so sind X und Y auf Null zu setzen. Die Werte von a, b, c, d, e und f stehen unter A, B, C, D, E und F bereits zur Verfügung.

10) Wie heißt die Funktion, deren Werte durch die nachstehende Ergibtanweisung berechnet werden können?

```
Y = ATAN (X/SQRT (1. - X*X))
```

11) Schreibe ein Programm, mit dessen Hilfe die ganze rationale Funktion 3. Grades (das Polynom)

$$f(x) = ax^3 + bx^2 + cx + d$$

durch den Ausdruck

$$x - r$$

dividiert werden kann. Unter r soll hierbei eine Lösung der Gleichung f(x) = 0 verstanden werden. Es kann vorausgesetzt werden, daß die Werte von a, b, c, d und r bereits gespeichert sind. Beachte, daß bei der Division kein Rest auftreten darf, da f(r) = 0 ist.

12) Kombiniere die Methoden von Aufg. 8) und Aufg. 11) so miteinander, daß es möglich ist, alle drei Lösungen einer kubischen Gleichung zu bestimmen. Nach Voraussetzung sollen alle drei Lösungen reell sein.

14.7. Eingabe und Ausgabe

Um in der FORTRAN-Sprache eine Eingabe- oder Ausgabeoperation anzugeben, muß man sich folgende Fragen vorlegen:

1) Welches Eingabe/Ausgabe-Gerät soll verwendet werden?
2) Für welche Variablen sind Werte zu lesen bzw. zu schreiben?
3) In welchem Format (reell, ganzzahlig usw.) liegen die Daten vor?
4) Auf welchen Stellen (z. B. auf welchen Kartenspalten) befinden sich die Daten im Eingabe- bzw. Ausgabesatz?

In der FORTRAN-Sprache werden bei Beantwortung dieser Fragen meist zwei verschiedene Statements zur Durchführung der Eingabe/Ausgabe-Operationen erforderlich sein. Mit einer FORMAT-Vereinbarung hat man das Format der Daten sowie die von ihnen eingenommenen Stellen innerhalb des Eingabe/Ausgabe-Satzes festzuhalten; in den READ- bzw. WRITE-Anweisungen hält man schließlich das ausgewählte Eingabe/Ausgabe-Gerät sowie die in die Operation eingehenden Variablen fest. Die Eingabeanweisung wird gewöhnlich in der nachstehenden Form verwendet:

```
READ (a,b) liste
```

Hierbei bedeutet a entweder eine ganzzahlige Konstante ohne Vorzeichen oder eine ganzzahlige Variable, die einen positiven Wert besitzen muß (Einheitennummer). Durch b wird die Statementnummer der zugehörigen FORMAT-Vereinbarung festgehalten, während die Datenliste liste aus einer Reihe von Variablennamen besteht. Die einzelnen in der Liste aufgeführten Namen sind durch Kommas voneinander zu trennen. Durch die READ-Anweisung wird bewirkt, daß für jede in der Liste enthaltenen Variablen ein Wert gelesen wird. Beispiele:

```
READ (5, 545) A, B, K, G18
READ (1, 56) C
READ (8, 89898) H7LP, LISTR
```

Mit dem für a eingesetzten Wert bezieht man sich auf ein Eingabe/Ausgabe-Gerät gemäß der folgenden Festlegungen:

a) Betriebssystem DOS
a=1 bezieht sich auf SYSIPT, a=2 auf SYSPCH, a=3 auf SYSLST und a=15 auf SYSLOG. Verwendet man die Nummern a=4 bis a=14, so bezieht man sich damit auf die Einheiten SYSnnn, wobei nnn=a-3 ist, d.h. a=4 bezieht sich auf SYS001, a=14 auf SYS011 usw. Werte für a, die größer als 15 sind, sind nicht erlaubt.

Ein Systemprogrammierer kann die Standardordnungen ändern, so daß a=1 bis a=4 den Einheiten SYS001 bis SYS004 entsprechen. Für a=5 kommt SYSIPT, für a=6 SYSLST und für a=7 schließlich SYSPCH zum Zuge. SYS005 bis SYS011 treten dann für die Werte a=8 bis a=14 ein, so wie zuvor schon geschildert.

b) Betriebssystem OS
Den Einheitennummern a=1 bis a=99 entsprechen die vom Kompilierer generierten internen Dateinamen (ddnames) FTxxFyy. Unter xx ist der Wert von a zu sehen, während yyy die laufende Nummer der Dateien bedeutet, die zur Einheit a gehören,

normalerweise also 001. Wenn die von der IBM unterstützten katalogisierten Prozeduren benutzt werden, so schließt der Ausführungsschritt (GO-Step) drei DD-Statements ein, die wie folgt aussehen:

```
//FTØ5FØØ1   DD   DDNAME=SYSIN
//FTØ6FØØ1   DD   SYSOUT=A
//FTØ7FØØ1   DD   SYSOUT=B
```

Jeder Systemprogrammierer ist in der Lage, die Prozeduren zu ändern, so daß beispielsweise die Einheitennummern 1,3 und 2 an die Stelle der Nummern 5,6 und 7 treten.

In der vollen FORTRAN-Sprache darf man die READ-Anweisung in folgender anderen Form benutzen:

READ (a,b,END=c,ERR=d) liste

Die Verwendung der beiden Angaben END=c bzw. ERR=d ist freigestellt, es darf sowohl die eine, als auch die andere weggelassen werden. Werden beide niedergeschrieben, so dürfen sie in einer beliebigen Reihenfolge notiert werden. Unter c und d sind jeweils Statementnummern zu verstehen. Stößt der Programmablauf auf das erste READ-Statement, nachdem der letzte Datensatz eingelesen wurde, so erfolgt eine Verzweigung zum Statement mit der Nummer c, wenn END=c in der READ-Anweisung aufgeführt ist. Ist ERR=d in die READ-Anweisung aufgenommen worden, so erfolgt ein Sprung zum Statement mit der Nummer d, wenn beim Lesen eines Datensatzes ein Fehler auftritt. Es folgen für diese erweiterte READ-Anweisung einige Beispiele:

```
READ (5,345,END=789)   A,B,K,G18
READ (5,345,ERR=78,END=789)   A,B,K,G18
READ (5,37,END=41,ERR=78)   HK,PUR
READ (5,37,ERR=78)   HK,PUR
```

Fehlt ERR=d, so bewirkt ein Lesefehler die anomale Beendigung des Jobsteps.

Die WRITE-Anweisung besitzt allgemein folgende Form:

WRITE (a,b) liste

Diese Anweisung unterscheidet sich von der READ-Anweisung lediglich dadurch, daß durch sie die Ausgabe von Werten bewirkt wird und nicht die Eingabe von Werten. Die beiden Optionen END und ERR sind in einer WRITE-Anweisung nicht zulässig.

Eine in einem FORTRAN-Programm angesprochene Datei wird normalerweise am Ende des Jobs abgeschlossen. Ein früheres Abschließen kann mittels der Anweisung

END FILE a

erzielt werden.

Die laufende Nummer der Datei wird durch dieses Statement um 1 erhöht. Der gleiche Effekt wird erzielt, wenn es zur Ausführung der Maßnahme kommt, die durch die END-Option eines READ-Statements angesprungen wird. Nachfolgende Anweisungen in einem Programm, die die gleiche Einheitennummer a wie die einer auf dieser Einheit abgeschlossenen Datei beinhalten, bewirken, daß die nächste Datenmenge auf der gleichen Einheit verarbeitet wird.

14.8. Die FORMAT-Vereinbarung

In diesem Abschnitt wollen wir uns hauptsächlich auf eine zu stanzende Karte beziehen, doch sollte von vornherein klargestellt werden, daß unsere Bemerkungen ebenso auch für alle anderen Eingabe/Ausgabe-Operationen gelten.

Die im WRITE-Statement aufgeführte Variablenliste (Datenliste) enthält die Namen der Variablen, deren Werte zu stanzen sind. Außerdem wird durch die Reihenfolge der Namen in der Liste auch festgelegt, in welcher Reihenfolge die Werte auf der Lochkarte erscheinen sollen. Die FORMAT-Vereinbarung betrifft die gleiche Karte, doch wird durch sie erklärt, wie die Werte auf die 80 Kartenspalten verteilt werden sollen. Die einfachste Form des FORMAT-Statements ist die folgende:

$$\underline{n}\ \text{FORMAT}\ (\underline{c_1}, \underline{c_2}, \cdots, \underline{c_n})$$

Unter $\underline{n}$ ist die Statementnummer dieser Vereinbarung zu verstehen; mit den c_i sind dagegen Formatangaben gemeint. Jeder Variablen in der Liste des WRITE-Statements entspricht eine und genau eine Formatangabe. Für jedes Kartenfeld legt die zugehörige Angabe fest, wie der Wert in das Feld zu stellen ist, d.h. in welchem Typ (reell oder ganzzahlig), mit welcher Länge, d.h. wieviel Kartenspalten reserviert werden sollen, und mit wieviel Dezimalstellen (falls solche angegeben werden dürfen). In diesem Abschnitt wollen wir uns zunächst auf die vier grundlegenden Formatangaben beschränken.

a) Formatangabe I

Sollen die Werte ganzzahliger Variabler gestanzt werden, so ist die I-Formatangabe zu verwenden. Für jede ganzzahlige Variable in der Liste des WRITE-Statements muß eine solche Angabe existieren und umgekehrt kann keine I-Formatangabe in der FORMAT-Vereinbarung vorhanden sein, ohne daß in der Liste der WRITE-Anweisung eine ganzzahlige Variable auftaucht.
Die I-Formatangabe besitzt die Form

Iw

Hierbei bedeutet w die Länge des Kartenfeldes, d.h. wieviel Kartenspalten vom Wert der ganzzahligen Variablen belegt werden sollen. Z.B. wird durch I7 ein siebenstelliges Kartenfeld festgelegt. Man betrachte die folgenden Beispiele:

Formatangabe	Wert der Variablen	Ausgabe [1]
I8	+1234	bbbb1234
I8	-1234	bbb-1234
I8	-37	bbbbb-37
I8	0	bbbbbbbØ
I3	-37	-37
I3	+37	b37
I3	768	768
I3	-768	***(Fehler)
I3	+1234	***(Fehler)
I3	0	bbØ
I3	+2	bb2

1) Durch b wird das Leerzeichen angezeigt.

Folgende Regeln gelten für I-Formatangaben:

1) Die auszugebende Zahl wird rechtsbündig in das Ausgabefeld gestellt.
2) Ist das Feld zu lang, um alle Ziffern und evtl. das Vorzeichen des Ausgabewertes aufnehmen zu können, so wird das Feld nach links mit Leerzeichen aufgefüllt.
3) Ist der Ausgabewert negativ, so wird unmittelbar vor die erste gültige Ziffer ein Minuszeichen eingefügt.
4) Führende Nullen werden grundsätzlich durch Leerzeichen ersetzt; ist der Ausgabewert gleich Null, so erscheint allerdings in der Einerstelle des Feldes die Ziffer 0.
5) Ist der Ausgabewert zu groß, d.h. erfordert er die Belegung von mehr Stellen als die Formatangabe vorsieht, so werden Sternzeichen in das Ausgabefeld gestellt. Dadurch kommt es zur Ausgabe einer fehlerhaften Information, wenn das Objektprogramm abläuft. Man sollte sich deshalb immer vor Augen halten, daß ein Feld, das lang genug ist, um bestimmte Werte einer Variablen aufnehmen zu können, durchaus nicht lang genug zu sein braucht, um alle möglichen Werte der Variablen aufnehmen zu können. Somit ist es unmöglich, daß der Kompilierer Formatangaben entdecken kann, die nicht genügend Platz für eine Variable vorsehen.

   In zweifelhaften Fällen sollte man deshalb lieber mehr Platz für eine Variable reservieren als tatsächlich erforderlich ist. Nebenbei bemerkt wird durch zusätzlich entstandene Leerstellen zwischen aneinandergrenzenden Felder die Lesbarkeit der Ausgabewerte erheblich verbessert.

b) Formatangabe E

Die Formatangabe E wird dazu verwendet, um die Werte reeller Variabler zu stanzen. Allgemein lautet ihre Form:

Ew.d

Durch w wird die Länge des Ausgabefeldes festgelegt, durch d die Anzahl der Dezimalstellen. Es ist notwendig, daß stets die Relation w>d+6 eingehalten wird, anderenfalls wird bereits während der Übersetzung vom Kompilierer ein Fehler festgestellt. Man betrachte dazu die folgenden Beispiele:

Formatangabe	Wert der Variablen	Ausgabe [1]
E15.8	+1234.567	b0.12345670Eb04
E15.8	- 1234.567	-0.12345670Eb04
E15.8	.00087	b0.87000000E-03
E15.4	+ 12345567	bbbbb0.1235Eb04
E15.4	-.00087	bbbb-0.8700E-03
E15.4	0.0	bbbbb0.0000Eb00
E11.4	-.00087	-0.8700E-03

Für die E-Formatangaben gelten folgende Regeln:

1) Der Exponent belegt stets ein aus vier Spalten bestehendes Teilfeld. In diesen vier Spalten steht zunächst ganz links der Buchstabe E, danach folgt bei negativen Exponenten ein Minuszeichen, sonst ein Leerzeichen, und die rechten beiden Stellen des Exponentenfeldes werden vom zweiziffrigen Exponenten selbst in Anspruch genommen.

1) b bedeutet hier das Leerzeichen.

2) Der Ausgabewert wird rechtsbündig ins Ausgabefeld gestellt, Leerzeichen werden nach links ergänzt, falls w>d+7 ist.

3) Wenn notwendig, wird die Zahl nach rechts (d.h. die Mantisse) durch Nullen ergänzt.

4) Ist eine unzureichende Anzahl von Dezimalstellen angegeben, so wird die auszugebende Zahl auf die letzte angegebene Dezimalstelle gerundet.

c) Formatangabe D [1]

Die Formatangabe D wird zusammen mit Variablen verwendet, deren Werte erhöht genau sind. An Stelle des E wird der Buchstabe D gestanzt. Ansonsten gilt für diese Formatangabe genau dasselbe, was bereits über die Formatangabe E gesagt wurde. Betrachten wir noch einige Beispiele:

Formatangabe	Wert der Variablen	Ausgabe [2]
D15.8	+1234.567	b0.12345670Db04
D15.8	-.00087	-0.87000000D-03
D15.6	9.	bbb0.900000Db01

d) Formatangabe F

Die Formatangabe F ist eine Alternative zu den Formatangaben E bzw. D und kann deshalb sowohl für reelle als auch für erhöht genaue Variablen angewendet werden. Ihre allgemeine Form lautet:

Fw.d

Hierbei bedeutet w die Länge des Feldes, und durch d wird die Anzahl der Dezimalstellen angezeigt, die im Ausgabefeld zur Verfügung stehen sollen. Grundsätzlich muß darauf geachtet werden, daß w>d für positive Zahlen und w>d+1 für negative Zahlen ist, sollen nicht Fehler festgestellt werden. Man betrachte folgende Beispiele:

Formatangabe	Wert der Variablen	Ausgabe [2]
F15.8	1234.567	bb1234.56700000
F15.8	.000123876	bbbbb0.00012388
F8.6	000123876	0.000124
F8.6	3.4	3.400000
F8.6	546.37	******** (Fehler)
F8.6	-3.4	******** (Fehler)
F10.6	3.4	bb3.400000

Für die Verwendung der Formatangabe F gelten folgende Regeln:

1) Die im Hauptspeicher stehende Zahl wird in eine Dezimalzahl umgewandelt, die d Dezimalstellen hat und die vor dem Dezimalpunkt w-d-1 Stellen aufweist.

2) Ist die Zahl negativ, so wird ein Minuszeichen unmittelbar vor die erste gültige Ziffer eingefügt.

---

1) Nur für die volle FORTRAN-Sprache definiert.

2) b bedeutet hier das Leerzeichen.

3) Führende Nullen vor der Einerstelle werden durch Leerzeichen ersetzt.

4) Wenn notwendig, wird die Zahl nach rechts durch Nullen ergänzt.

5) Sind für die auszugebende Zahl zu wenig Stellen vorgesehen, so wird die Zahl im Feld durch Sternzeichen ersetzt; somit wird beim Ablauf des Objektprogrammes eine Fehlernachricht ausgegeben.

Der Vorteil der Formatangabe F besteht darin, daß das von ihr produzierte Ausgabefeld leichter zu lesen ist und darüberhinaus weniger Stellen erfordert als gleichwertige E- oder D-Formatangaben. Nachteilig ist jedoch, daß der Programmierer den maximalen Absolutwert der auszugebenden Zahlen kennen muß, wenn er Fehler vermeiden will.

Zur besseren Illustration der Formatangaben soll folgendes Beispiel dienen:

```
128 FORMAT (15,F7.2,D18.8,I7)
    WRITE (6,128)  K,A,Q,M
```

Nehmen wir weiterhin an, daß die Variablen z. Z. der Ausgabe-Anweisung folgende Werte besitzen: K=345, A=-17.890022, Q=0.0031416282 und M=-8. Es wird dann folgende Karte gestanzt:

```
bb345b-17.89bbbbØ.31416282D-Ø2bbbbb-8
```
[1)]

Verwenden wir jetzt an Stelle des o.a. WRITE-Statements das Statement

```
WRITE (6,128)  M,Q,A,K
```

so erhalten wir folgende Karte:

```
bbb-8bbbØ.ØØbbb-Ø.1789ØØ22DbØ2bbbb345
```
[1)]

Schreiben wir aber die Anweisung

```
WRITE (6,128)  K,M,A,Q
```

so erhalten wir eine Fehlernachricht, weil auf die zweite und auf die vierte Variable in der Liste eine Formatangabe bezogen ist, die nicht mit dem Typ der Variablen übereinstimmt.

14.9. Format von Eingabedaten[2)]

Eingabedaten werden meist in der gleichen Form in die Karten gelocht wie sie durch ein FORTRAN-Objektprogramm in Karten gestanzt werden. Es sind jedoch bestimmte Erleichterungen gestattet, um die Vorbereitung von Eingabedaten bequemer zu gestalten[1)].

1) Es ist nicht notwendig, bei den Formatangaben D und E vier Spalten für die Aufnahme des Exponentenfeldes vorzusehen; der Exponent braucht nicht in seiner regulären Form abgelocht zu werden. Statt der Ablochung von beispielsweise E+02 genügt auch Eb02 oder E02 oder E2 oder +02 oder sogar einfach +2. An Stelle von E-02 sind auch die Formen E-2, -02 und -2 erlaubt. Kurz zusammengefaßt, bei Exponenten kann man auf das Ablochen führender Nullen verzichten, ebenso wie auf das Ablochen eines Pluszeichens. Darüberhinaus kann man auch auf das Ablochen der Buchstaben E bzw. D verzichten,doch muß dann auf jeden Fall das Vorzeichen des Exponenten abgelocht werden;

1) b bedeutet hier das Leerzeichen.

2) Die Formatangabe D existiert auch bei der Eingabe nur für die volle Sprache. Erscheint bei Beispielen von Eingabedaten das Zeichen b, so ist damit das Leerzeichen gemeint.

2) Der Dezimalpunkt kann weggelassen werden, wenn Eingabedaten für E-, D- bzw F-Formatangaben präpariert werden. Wird auf das Ablochen des Punktes verzichtet, so unterstützt das Objektprogramm die Anzahl von Dezimalstellen, die in der Formatangabe vorgeschrieben wurden. So wird z.B. bei Zugrundelegung der Formatangabe F7.4 eine Zahl, die zu bbb8734 abgelocht wurde, so behandelt, als ob sie zu bb.8734 abgelocht worden wäre.

3) Bei allen Feldern darf ein Minuszeichen irgendwo ins Feld hinein gestellt werden; Hauptsache es erscheint links vor der ersten von Null verschiedenen Ziffer der Zahl bzw. links vom Dezimalpunkt. Somit wird b-b78 genau so behandelt wie bb-78.

4) Eine Zahl kann mitten in ein Feld hineingestellt werden, d.h. sie kann von Leerzeichen umgeben sein [1]; diese werden als Nullen angesehen, falls es die Formatangabe erfordert. Beispielsweise wird bb667b als 66.70 behandelt, wenn die Formatangabe F6.2 vorliegt. Ein Feld, das lauter Leerzeichen enthält, wird so behandelt, als ob der Wert 0 abgelocht wäre.

5) Wird in Feldern, die durch F-, E- oder D-Formatangaben festgelegt sind, ein Dezimalpunkt gelocht, so überschreibt seine Position die Formatangabe. Somit wird bb7.4 bei Vorliegen der Formatangabe F5.3 nicht anders behandelt, als wenn für diesen Wert die Formatangabe F5.1 vorgelegen hätte.

14.10. Ein Beispiel

Wir haben bis zu diesem Punkt über die Grundzüge der FORTRAN-Sprache berichtet.

Wir sollten deshalb auch jetzt schon in der Lage sein, ein leidlich verwickeltes Programm zu schreiben. Zur Demonstration wollen wir uns deshalb mit Problem der Amortisierung von Hypotheken befassen. Wir wollen vorsehen, daß der Computer Karten einlesen soll, auf denen drei verschiedene Daten abgelocht sind. Es sind dies: Höhe der Hypothek, der jährliche Zinsfuß der Hypothek und der monatliche Ratenzahlungsbetrag. Wir beabsichtigen, für die betreffende Hypothek einen Tilgungsplan aufzustellen, der für jeden Monat der Laufzeit der Hypothek ausweist, welcher Teilbetrag der monatlichen Ratenzahlung zum Ausgleich der angefallenen Zinsen dient, welcher Teilbetrag zur Tilgung der Schuld beiträgt und wie hoch die restliche Hypothekenschuld ist. Wie gewöhnlich entwerfen wir uns zunächst einmal einen Programmablaufplan (Abb. 14.3.).

1) Wir erinnern uns, daß bei PL/1 keine rechtsbündigen Leerzeichen als Nullen angesehen werden.

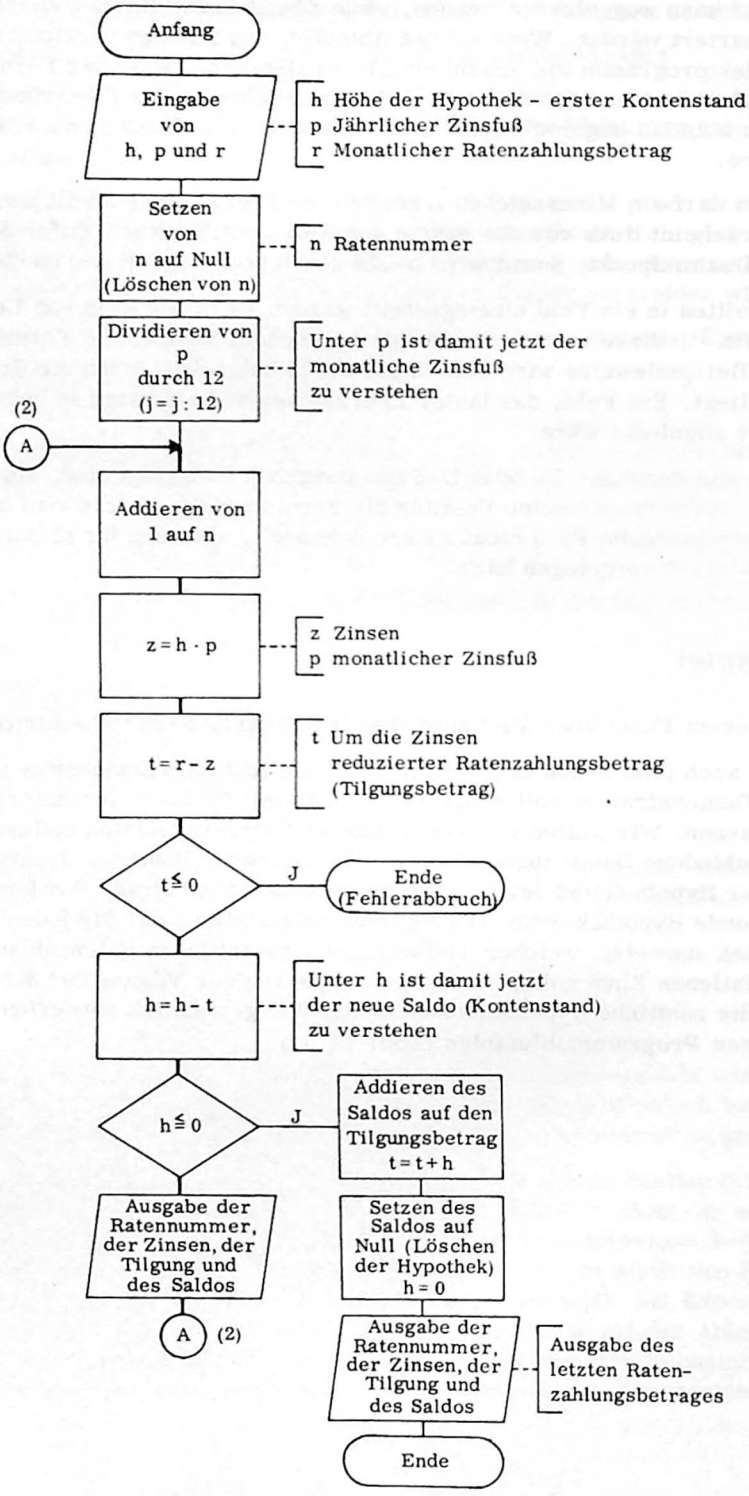


Abb. 14.3. Programmablaufplan

Auf Grund dieses Planes sind wir in der Lage, ein Programm zur Lösung dieser Aufgabenstellung zu schreiben (Abb. 14.4.).

```
      READ (1,1Ø1)  HYP,PROZ,RATE
      N = Ø
      PROZ = PROZ/12
    2 N = N + 1
      ZINS = PROZ*HYP
      TIL = RATE - ZINS
      IF (TIL)  3,3,4
    3 STOP
    4 HYP = HYP - TIL
      IF (HYP) 5,5,6
    6 WRITE (3,3Ø1)  N,ZINS,TIL,HYP
      GO TO 2
    5 TIL = TIL + HYP
      HYP = Ø.Ø
      WRITE (3,3Ø1)  N,ZINS,TIL,HYP
      STOP
  1Ø1 FORMAT (F1Ø.2,F1Ø.8,F1Ø.2)
  3Ø1 FORMAT (I5,F1Ø.2,F1Ø.2,F1Ø.2)
      END
```

Abb. 14.4. Programm zur Aufstellung eines Hypothekentilgungsplanes

Nehmen wir an, daß die erste Eingabekarte wie folgt abgelocht ist:

```
bb12ØØØ.ØØØ.Ø6ØØØØØØbbbb18Ø.ØØ
```
1)

Als die ersten beiden Ausgabezeilen werden dann die folgenden Zeilen gedruckt:

```
bbbb1bbbbb6Ø.ØØbbbb12Ø.ØØbb1188Ø.ØØ
    2      59.4Ø     12Ø.6Ø  11759.4Ø
```
1)

Die beiden letzten Ausgabezeilen besitzen dann folgendes Aussehen:

```
bbb81bbbbbb1.16bbbb178.84bbbbb52.9Ø
   82       Ø.26      52.9Ø       Ø.ØØ
```
1)

Aufgabengruppe 38

1) Schreibe ein FORTRAN-Programm zur Berechnung der Reihe

$$\arcsin(x) = x + \frac{1}{2}\cdot\frac{x^3}{3} + \frac{1}{2}\cdot\frac{3}{4}\cdot\frac{x^5}{5} + \frac{1}{2}\cdot\frac{3}{4}\cdot\frac{5}{6}\cdot\frac{x^7}{7} + \ldots$$

im Intervall [0, 1]. Als Schrittweite für die x-Werte ist h= 0,05 zu wählen. Es ist eine Tabelle zu drucken; in der ersten Spalte sollen die x-Werte, in der zweiten die arcsin(x)-Werte erscheinen. Beachte, daß arcsin(0) = 0 und arcsin(1) = 1,5708 ist. Zur Berechnung sind so viele Glieder der Reihenentwicklung heranzuziehen, daß die Ergebnisse bis zur vierten Dezimalstelle exakt sind.

1) b bedeutet hier das Leerzeichen.

Hinweis:

$$a_{n+1} = a_n \frac{(2n-3)^2 x^2}{(2n-2)(2n-1)}$$

2) Schreibe ein Programm, mit dem die Quadratwurzeln der ganzen Zahlen von 10 bis einschließlich 99 berechnet werden können. Die Ausgabe soll auf Karten erfolgen. Pro Karte ist ein Wertepaar zu stanzen:

Spalten 1 bis 3: Radikand im Format I3
Spalten 4 bis 21: Quadratwurzel im Format F18.7

3) Der Mittelwert $\overline{X}$ einer Menge von Zahlen x ist definiert als Summe dieser Zahlen dividiert durch die Anzahl n der Zahlen

$$\overline{X} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$$

Für die Varianz s^2 der Zahlenmenge gilt folgende Formel

$$s^2 = \frac{1}{n \cdot (n-1)} \left[n \cdot \sum_{i=1}^{n} (x_i)^2 - \left(\sum_{i=1}^{n} x_i \right)^2 \right]$$

Unter der Standardabweichung s versteht man schließlich

$$s = \sqrt{s^2}$$

Schreibe ein Programm, das eine beliebige Anzahl von Karten, jedoch weniger als 10000, einlesen kann; diese enthalten auf den Spalten 1 bis 3 die x_i-Werte (ganze Zahlen). Die letzte Karte kann dadurch identifiziert werden, daß bei ihr zusätzlich -30 auf den Spalten 4 bis 6 abgelocht ist, während bei den anderen Karten diese Spalten leer bleiben.

Die Werte von n, $\overline{X}$ und s sollen auf einer Karte gestanzt werden.

4) Schreibe ein Programm zur Berechnung von H_n; H_n ist definiert zu

$$H_n = \sum_{k=1}^{n} \left[\frac{1}{k} - \ln(n) \right] \qquad \text{für } n = 1, 2, \cdots, 9, 10, 20, 30$$

Jedes Wertpaar von n und H_n ist auf einer Karte zu stanzen. Für alle n gilt

$$H_n \leqq 1$$

(Zu bemerken ist, daß die Rundungsfehler bei einigen dieser Berechnungen zu sehr großen Werten auflaufen können).

5) Schreibe ein Programm, das die Koeffizienten quadratischer Gleichungen in der Normalform $ax^2 + bx + c = 0$ einliest und die Lösungen auf Karten ausgibt. Es kann vorausgesetzt werden, daß beide Lösungen reell sind. Auf den Eingabekarten sind die Koeffizienten a, b und c jeweils im Format E18.8 abgelocht.

6) Das Newtonsche Näherungsverfahren zur Bestimmung der Lösung einer Gleichung basiert auf der Formel

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Hierbei bedeutet $f'(x_n)$ die Ableitung der Funktion $f(x)$ an der Stelle $x = x_n$. Die Rechnung soll mit $x_0 = 1$ beginnen, daraus kann x_1 berechnet werden. Aus x_1 läßt sich x_2 ermitteln usw. Das Verfahren ist solange fortzusetzen, bis die Differenz zweier aufeinanderfolgender x-Werte absolut kleiner als ein vorgegebener Fehler h ist.

Dieses Verfahren soll in einem Prog ramm zur Bestimmung einer Lösung der kubischen Gleichung $ax^3 + bx^2 + cx + d = 0$ verwendet werden. Zur Eingabe gelangen Lochkarten; auf je einer Karte sind die Koeffizienten a, b, c und d jeweils im Format E18.8 abgelocht. Die gefundene Lösung soll im gleichen Format in eine Karte gestanzt werden. Für dieses Polynom ist die Ableitung $f'(x) = 3ax^2 + 2bx + c$.

7) Die Koeffizienten einer quadratischen Gleichung in der Normalform $ax^2 + bx + c = 0$ sind wie folgt im Format E12.5 auf Karten abgelocht:

Spalten	Inhalt
1 bis 12	Realteil von a
13 bis 24	Imaginärteil von a
25 bis 36	Realteil von b
37 bis 48	Imaginärteil von b
49 bis 60	Realteil von c
61 bis 72	Imaginärteil von c

Die beiden komplexen Lösungen x_1 und x_2 sind auf eine Karte im Format E12.5 wie folgt zu stanzen:

Spalten	Inhalt
1 bis 12	Realteil von x_1
13 bis 24	Imaginärteil von x_1
25 bis 36	Realteil von x_2
37 bis 48	Imaginärteil von x_2

8) Schreibe ein Programm zur Berechnung des Umfanges von Dreiecken. Die Eingabe der Seitenlängen des Dreiecks soll in Form dreistelliger ganzer Zahlen über die Konsole erfolgen können.

Die Ausgabe des Ergebnisses erfolgt ebenfalls über die Konsole. Bilden die drei Seiten kein Dreieck, so soll statt des Umfanges die Zahl 0 geschrieben werden.

9) Das Programm zu Aufgabe 8) ist so abzuändern, daß außer des Umfanges auch noch der Flächeninhalt des Dreieckes ausgegeben wird.

Zur Berechnung des Flächeninhaltes soll die folgende Formel verwendet werden:

$$s = \sqrt{s\,(s-a)\,(s-b)\,(s-c)}$$

Hier bedeuten a, b und c die Seitenlängen und s ist gleich dem halben Umfang.

10) Es soll folgendes Gleichungssystem von zwei Gleichungen mit zwei Unbekannten gelöst werden:

I. $ax + by = c$
II. $dx + ey = f$

Alle Koeffizienten sind auf einer Karte jeweils im Format E12.6 abgelocht. Die Lösungen sind auf einer Lochkarte auszugebe; ist eine eindeutige Lösung vorhanden, so sind die Werte für x und y jeweils im Format E14.6 zu stanzen. Bei nicht eindeutigen Lösungen soll keine Karte gestanzt werden, dagegen sind alle Koeffizienten sowie der Wert **0** für x und für y über die Konsole anzuzeigen.

11) Aufgabe 10) ist so abzuändern, daß die x- und y-Werte im Format F14.6 gestanzt werden, wenn die beiden Lösungen im Wertebereich von -99,999996 bis 99,999999 liegen, anderenfalls ist die Formatangabe zu verwenden, die bei Aufgabe 10) gefordert ist.

12) Zur Berechnung von bestimmten Integralen wird in der Praxis vielfach die Regel von Simpson verwendet.

Es gilt

$$F = \int_a^{a+2h} f(x)\,dx = \frac{h}{3}\,[f(a) + 4f(a+h) + f(a+2h)]$$

Es soll nun ein Programm zur Berechnung von

$$F = \int_1^4 x^4\,dx$$

geschrieben werden (a = 1, h = 1).

Will man eine größere Genauigkeit erhalten, so muß man für h einen kleineren Wert wählen, in den man das Intervall in kleinere Abschnitte unterteilt. Hieraus ergibt sich folgende allgemeingültige Formel. wobei a und b die Abszissenwerte der Intervallendpunkte darstellen.

$$F = \int_a^b f(x)\,dx$$

$$= \frac{h}{3}\,[f(a) + 4\,f(a+h) + 2\,f(a+h) + 4\,f(a+3h) + 2\,f(a+4h) + \cdots$$

$$+ 2\,f(b-2h) + 4\,f(b-h) + f(b)]$$

Hier ist b = a + 2hn, wobei n eine beliebige ganze Zahl ist. Für n = 1 ergibt sich die Ausgangsformel.

Verwende n = 2, 3, 4. Daraus ergeben sich für h der Reihe nach die Werte $\frac{1}{2}$, $\frac{1}{3}$ und $\frac{1}{4}$ bei der Berechnung von

$$F = \int_1^3 x^4\,dx$$

Wie beeinflußt die Vergrößerung von n die Genauigkeit des Resultates? (Für das Integral beträgt der exakte Wert 48,4).

13) Eine gewisse Bedeutung besitzt das Integral

$$\int_0^z \exp\left(\frac{t^2}{2}\right)\,dt$$

Hierbei bedeutet z irgendeine positive Zahl. Schreibe ein Programm, mit dessen Hilfe dieses Integral auf drei Dezimalstellen genau berechnet werden kann. Für z sind zunächst der Reihe nach die Werte 0.1, 0.2, ···, 2.0 zu wählen. Die Berechnung ist mit folgenden Werten von z fortzusetzen:

2.5, 3.0, 3.5, 4.0, 5.0, 10.0, 20.0 und 40.0.

Welche Aussagen kann über den Wert des Integrals getroffen werden, wenn der Wert von z gegen Unendlich geht?

14) Zur Berechnung des Integrals

$$\int_0^{\infty} \frac{dx}{x}$$

ist die Simpsonsche Regel einzusetzen. Bei der Berechnung sind so viele Glieder zu nehmen, bis ein genügend kleines Glied auftritt. Da nur eine begrenzte Anzahl von Gliedern verwendet wird, entsteht ein Abbrechfehler.

Es ist ein Programm zu entwickeln, das so viele Glieder zur Berechnung verwendet, bis ein letztes Glied gefunden wird, das kleiner als a) 10^{-4}, b) 10^{-5}, c) 10^{-6} ist.

15) Entwickle eine auf zwei Dezimalstellen genaue Lösung für folgendes Gleichungssystem

$$\text{I.} \quad \sin y = 4x + 3$$
$$\text{II.} \quad \cos x = 7y + 8$$

16) Schreibe ein FORTRAN-Programm zur Lösung der Gleichung

$$b^x = a$$

Die Werte von a und b sind einzulesen. Zu drucken oder zu stanzen sind a, b und die Lösung x.

17) Schreibe ein FORTRAN-Programm, das einen Wert für T einliest und das den Wert von T sowie den Wert von x, der die Beziehung

$$T = \int_0^x \exp(0,5\,t^2)\,dt$$

erfüllt, ausdruckt oder stanzt. Gleichungen dieser Art werden als Integralgleichungen bezeichnet. Die Berechnung soll so erfolgen, daß x bis auf zwei Dezimalstellen genau ist.

18) Ändere das Programm zu Aufgabe 17) so ab, daß x so genau berechnet wird, wie immer es bei Verwendung der Simpsonschen Regel und n = 1, 2, 4, 8, 16 möglich ist. Bei der Ausgabe sind neben den x-Werten auch die Werte von n aufzuführen. Verfolge, wie die Programmablaufzeit anwächst, wenn n ständig größer wird.

15. Zusätzliche FORTRAN-Statements

In den beiden Betriebssystemen DOS und OS stehen Kompilierer für die volle FORTRAN-Sprache zur Verfügung. Wir hätten es deshalb eigentlich nicht nötig, besonders zu betonen, welche Sprachelemente über BASIC FORTRAN hinausgehen. Wenn wir es dennoch tun, so tun wir es im Interesse der Kompatibilität und des dadurch möglichen Programmaustausches auch mit Computern, deren Software nur einen Kompilierer für BASIC FORTRAN einschließt.

15.1. Alphanumerische Formatangaben

Neben Formatangaben für numerische Felder sind auch solche zur Behandlung alphanumerischer Informationen vorgesehen. In der FORTRAN-Sprache werden alphanumerische Daten hauptsächlich für Überschriften und für Kennzeichnungen (Niederlegung von Stichpunkten) eingesetzt, in Verbindung mit der IF-Anweisung darüber hinaus auch zur Prüfung von alphanumerischen Feldern auf Gleichheit.

1) Formatangaben vom H-Typ

Hollerithangaben besitzen die Form

wH,

wobei dieser Angabe genau w alphanumerische Zeichen folgen müssen. Bei der IBM /370 muß w kleiner als 256 sein. Die dem Buchstaben H unmittelbar folgenden w Zeichen (die alle gültigen IBM /370-Verschlüsselungen einschließlich des Leerzeichens umfassen dürfen) werden aus der FORMAT-Vereinbarung herausgezogen und in den Ausgabesatz eingefügt. Bei der Eingabe werden w Zeichen aus dem Eingabesatz gelesen; diese ersetzen die im FORMAT-Statement dem Buchstaben H unmittelbar folgenden Zeichen. In Abb. 15.1. finden wir hierzu ein kleines Beispiel.

```
1Ø FORMAT (3H X=,F12.6,4H  Y=,F12.6)
11 FORMAT (F12.6,F12.6)
12 READ (1,11)  X,Y
   X = X + Y
   WRITE (3,1Ø)  X,Y
   GO TO 12
   END
```

Abb. 15.1. Beispiel für die H-Formatangabe

Bei diesem Beispiel möge es zu folgender Ausgabe kommen:

```
bX=bbb12.678223bbY=bb-13.ØØ1477 1)
 X=     Ø.ØØØØØØ  Y=     Ø.ØØØØØØ
```

Wir ändern nun das READ-Statement dieses Beispieles wie folgt ab:

```
READ(5,1Ø) X,Y
```

1) Durch b wird im Abschnitt 1.5.1. das Leerzeichen angezeigt.

Die einzulesenden Karten mögen jetzt die folgenden Zeichen enthalten:

```
bX=bbbb1.Ø6ØØØØbbZ=bbb-Ø.18ØØØØ
Xb=bbbbbbbbbbbbbbY=bbbb2187ØØØØ
```

Unter diesen Voraussetzungen kommt es zu der nachstehenden Ausgabe

```
bX=bbb-Ø.88ØØØØbbZ=bbb-Ø.18ØØØØ
Xb=bbb21.87ØØØØbbY=bbb21.87ØØØØ
```

H-Formatangaben werden hauptsächlich für die Ausgabe von Überschriften eingesetzt.

Hollerithfelder unterscheiden sich hierbei in folgender Hinsicht von numerischen Feldern:

a) Bei numerischen Feldern werden Leerzeichen nicht beachtet, während sie hier als Zeichen betrachtet werden, die den anderen gleichwertig sind.

b) Bei numerischen Feldern wird der auszugebende Wert über ein Element der Datenliste angesprochen, d.h. durch den Namen einer Variablen; bei Hollerithfeldern wird die auszugebende Information unmittelbar in das FORMAT-Statement geschrieben.

c) Da für ein Hollerithfeld kein Variablenname in der Datenliste existiert, kann auf das betreffende Feld eine Bezugnahme nur über die gleiche FORMAT-Vereinbarung erfolgen.

2) Formatangabe vom X-Typ

Ihre allgemeine Form lautet:

wX

Hierbei muß beim System IBM /370 für w eine ganze Dezimalzahl kleiner als 256 angegeben werden. Bei der Eingabe werden w aufeinanderfolgende Stellen auf dem Eingabemedium ignoriert, bei der Ausgabe werden w Leerzeichen in den Ausgabesatz gestellt. Im allgemeinen verwendet man Formatangaben vom X-Typ dazu, bei der Eingabe Felder zu überspringen, die vom Programm nicht benutzt werden. Bei der Ausgabe dienen sie hauptsächlich dazu, Leerstellen im Ausgabesatz zu erzeugen (bessere Lesbarkeit von Überschriften usw.).

3) Literalangaben

Hollerithfelder können einfach dadurch festgelegt werden, daß man alphanumerische Informationen in Anführungsstriche einschließt. Man kann sie also an Stelle von H-Formatangaben schreiben. Beispiel:

```
FORMAT ('bX=',F12.6,'bbY=',F12.6)
```

Es sollte hierzu bemerkt werden, daß man bei der H-Formatangabe niederschreiben muß, wieviele Zeichen zum Hollerithfeld gehören; bei Literalangaben kann man sich das ersparen.

Ein Anführungszeichen kann in ein Hollerithfeld nur aufgenommen werden, wenn man zwei aufeinanderfolgende Anführungszeichen schreibt. Somit produziert die Angabe 'WENN''S' im Ausgabesatz die Zeichenfolge WENN'S.

4) Vorschubsteuerung

Wird zur Ausgabe ein Zeilendrucker benutzt, so wird der Inhalt des ersten Bytes nicht gedruckt. Es dient stattdessen zur Steuerung des Vorschubs. Das Steuerzeichen wird meist mittels einer H- oder einer Literalangabe festgelegt, doch ist für diesen Zweck auch jeder andere Typ einer Formatangabe erlaubt. Für Steuerungszwecke sind die folgenden Zeichen definiert:

b	Einfacher Zeilentransport vor dem Drucken
0	Doppelter Zeilentransport vor dem Druck
+	Kein Zeilentransport vor dem Drucken
1	Vorschub bis zur ersten Zeile der nächsten Seite (Sofortvorschub)

Erfolgt die für einen Zeilendrucker bestimmte Ausgabe zunächst auf ein Magnetband oder auf einen anderen Datenträger, dann wird das Steuerzeichen genau so geschrieben wie die anderen Datenzeichen auch.

5) Formatangaben vom A-Typ

Zur Verarbeitung alphanumerischer Daten ist die A-Formatangabe vorgesehen. Allgemein sieht sie wie folgt aus:

Aw

Bei der Eingabe werden durch diese Formatangabe w Zeichen aus dem Eingabesatz gelesen und ohne Umwandlung der Variablen zugewiesen, die in der Datenliste erscheint. Im allgemeinen wird w nicht größer sein als die Anzahl l der Bytes (2,4 oder 8), die der Variablen zugeordnet sind. Wenn w kleiner als die Bytezahl ist, so werden die alphanumerischen Daten linksbündig in das Feld gestellt. Nach rechts wird das Feld mit Leerzeichen aufgefüllt. Wenn w dagegen größer als die Variablenlänge l ist, so werden nur die l rechten Zeichen der Eingabezeichenfolge gespeichert, die ersten (linken) w-1 Zeichen gehen verloren. Ist bei der Ausgabe w kleiner als l, so werden nur die ersten l Zeichen ausgegeben. Ist w dagegen größer als l, so werden die Zeichen rechtsbündig in den Ausgabesatz gestellt, auf die ersten w-1 Stellen werden Leerzeichen gebracht.

Wenn die A-Formatangabe verwendet wird, so werden die Daten intern im EBCDI-Code gespeichert. Wird die Variable anschließend in einer arithmetischen Ergibtanweisung verwendet, so erfolgen die Berechnungen so, als ob die Daten als Dualzahlen vorliegen würden.

Wird beispielsweise also C342 eingelesen und der Variablen J (vom ganzzahligen Typ) zugewiesen, so enthält J, hexadezimal gesehen, den Wert C3F3F4F2. Bei arithmetischen Operationen ist dieser Wert somit äquivalent der Dezimalzahl - 1 007 422 222. Arithmetische Operationen mit alphanumerischen Daten werden hauptsächlich innerhalb von IF-Anweisungen verwendet, um festzustellen, welches von zwei alphanumerischen Feldern den größeren Inhalt aufweist. Man sollte jedoch gründliche Überlegungen anstellen, ob der gewünschte Effekt erreicht wird, wenn man plant, in eine arithmetische Operation sowohl alphanumerische Daten als auch arithmetische Daten eingehen zu lassen.

Um die normale Verwendung von A-Formatangaben zu zeigen, soll in Abb. 15.2. ein einfaches Beispiel betrachtet werden.

```
13 FORMAT (A3,1X,A3,I5)
12 FORMAT (1X,A3,' = ',I5)
 4 READ (1,13)  P,Q,J
   IF  (Q)  6,7,6
 7 STOP
 6 IF  (P-Q)  4,5,4
 5 WRITE (3,12)  P,J
   GO TO 4
   END
```

Abb. 15.2. Beispiel für die Verwendung von A-Formatangaben

Nehmen wir an, daß die drei ersten Datenkarten folgende Werte enthalten:

```
X71YX69b2174
X71YX7Øb8135
X71bX71b2143
```

Nach dem Einlesen der ersten Karte besitzen die Variablen die folgenden Werte:

Variable	Alphanumerische Zeichen	Sedezimalziffern
P	X7 1	E7 F7 F1 4 0
Q	X6 9	E7 F6 F9 4 9
J	(21 74) -numerisch	0 0 0 0 0 8 7 E

Da P-Q≠Ø ist, wird nichts geschrieben. Nur für die dritte Karte wird eine Zeile ausgegeben, die wir folgt aufgebaut ist:

```
X71b=bb2143
```

Beachte, daß bei der Eingabe der Wert, der J zugewiesen wird, in eine Dualzahl konvertiert wird und bei der Ausgabe wieder vom dualen ins dezimale Zahlensystem übertragen wird. Diese Konvertierungen erfolgen sogar dann, wenn keine Operation mit dem dualen J-Wert vorgesehen ist. Würde also für J auch eine A-Formatangabe vorliegen, so hätte man damit die Konvertierungen ausgeschieden.

15.2. Vielfache Formatangaben

Man stößt öfters auf FORMAT-Vereinbarungen, die sich wiederholende Formatangaben erfordern. Bei einem Statement der Form

```
FORMAT (F12.4,F12.4,F12.4,F18.7,F18.7)
```

z.B. begegnen wir drei aufeinanderfolgenden Formatangaben F12.4 und zwei Formatangaben F18.7. Dieses Statement kann wie folgt umgeschrieben werden:

```
FORMAT (3F12.4,2F18.7)
```

Allgemein können alle D-, E-, F-, I- oder A-Formatangaben so oft wiederholt werden, wie es gewünscht wird, indem man die Anzahl der Wiederholungen als ganze Dezimalzahl unmittelbar vor den betreffenden Buchstaben schreibt. Man muß nur darauf achtgeben, daß man nicht mehr als 121 (oder 133 oder 145) Zeichen für eine Druckzeile vorsieht und nicht mehr als 80 Zeichen für eine Karte. Die Wiederholungsfaktoren dürfen 255 nicht überschreiten.

Mitunter wird gewünscht, daß eine Leerkarte ausgegeben wird oder daß eine neue Druckzeile begonnen wird bzw. daß ein Eingabesatz übersprungen wird. Wir können das erreichen, indem wir einen Schrägstrich (/) als Formatangabe verwenden. Immer wenn im FORMAT-Statement ein Schrägstrich entdeckt wird, wird zu einem neuen Eingabesatz übergegangen bzw mit der Zusammenstellung eines neuen Ausgabesatzes begonnen. Man verfolge dazu die Beispiele

```
1234 FORMAT (/F12.4,F18.6)
1276 FORMAT (F12.4/F18.6)
```

Die Formatangaben im Statement 1234 bewirken z.B. bei der Kartenausgabe, daß eine Karte leer gelassen wird und daß anschließend die Werte zweier Variablen in die zweite

Karte gestanzt werden. Bei einer vorgesehenen Karteneingabe erreicht man durch diese Vereinbarung, daß eine Karte übergangen wird und daß von der darauffolgenden Karte zwei Werte eingelesen werden. Die bei der Vereinbarung 1276 vorgesehenen Formatangaben haben zur Folge, daß je ein Wert von den beiden Karten gelesen bzw. je ein Wert auf zwei Karten gestanzt wird.

Immer wenn FORMAT-Vereinbarungen für mehrere Sätze verwendet werden sollen, muß man sorgfältig darauf achten, daß auch wirklich jede Variablen mit dem Satz korrespondiert, in den ihr Wert hineingestellt bzw. aus dem ihr ein Wert zugewiesen werden soll.

Wird der Schrägstrich (/) codiert, so braucht nach dieser Angabe kein Komma geschrieben zu werden. Zwei oder mehr auf einanderfolgende Schrägstriche können dazu verwendet werden, um zusätzliche Leersätze auf das Ausgabemedium zu schreiben bzw. um Eingabesätze zu überspringen. Somit bewirkt z.B. das Statement

```
1284 FORMAT (F12.4///F18.6)
```

bei der Karteneingabe, daß für die erste Variable ein Wert von der ersten Karte gelesen wird und für die zweite Variable ein Wert von der vierten Karte.

Während der Eingabe/Ausgabe tastet das Objektprogramm die FORMAT-Vereinbarung und die Datenliste ab. Die Übertragung von Daten zwischen dem Hauptspeicher und dem EA-Gerät erfolgt solange, bis Daten zu oder von allen Elementen der Datenliste übertragen sind und entweder

a) das FORMAT-Statement abgearbeitet ist oder
b) die nächste Formatangabe zu den Formatangaben des numerischen Typs gehört (D, E, F, G, I oder A).

Die Regel schließt also ein, daß alle H-, X- und Literalformatangaben, die noch im FORMAT-Statement stehen, in die Ausgabeoperation eingehen, obwohl die Datenliste der WRITE-Anweisung bereits abgearbeitet ist. Erst danach wird die Ausgabe abgeschlossen. Die Statements

```
2 FORMAT (3HI=b,I4,E2Ø.8/7X,E2Ø.8)
  WRITE (7,2)  J,A,B
```

bewirken somit, daß zwei Karten gestanzt werden die beispielsweise folgende Informationen enthalten:

```
I=b|-234|bbbbb-Ø.78967832EbØ3
bbbbbbb|bbbbbbØ.34545656E-Ø1
```

Wird die obige WRITE-Anweisung durch

```
WRITE (7,2)  J,A
```

ersetzt, dann werden die beiden folgenden Karten ausgegeben:

```
I=b|-234|bbbbb-Ø.78967832EbØ3
bbbbbbb
```

Zu dieser Ausgabe kommt es, weil die Formatangaben bis zur E-Formatangabe abgearbeitet werden, obwohl keine weitere Variable in der Datenliste des WRITE-Statements erscheint.

Wird dagegen das WRITE-Statement

```
WRITE (7,2)  J,A,B,K
```

gegeben, so gehen die Formatangaben in der FORMAT-Vereinbarung eher zu Ende als die in der Datenliste aufgeführten Variablen. In solchen Fällen werden die Formatangaben vom

Anfang der FORMAT-Vereinbarung an wiederholt, so daß die folgenden Karten gestanzt werden:

```
I=b|-234|bbbbb-Ø.78967832Eb Ø3
bbbbbbb|bbbbbbØ.34545656E-Ø1
I=b|bb35
```

Mitunter ist es sinnvoll, statt einer einzigen Formatangabe eine ganze Gruppe von Formatangaben zu wiederholen. Man erreicht das, indem man die Formatangaben in Klammern einschließt und im Bedarfsfall außerdem einen Wiederholungsfaktor (ganze Dezimalzahl) vor die linke (geöffnete) Klammer schreibt. Man betrachte dazu folgendes Beispiel:

```
3 FORMAT (' ',2I5,2(F18.6,2F15.6))
  WRITE (6,3)  JL,JM,A1,A2,A3,B1,B2,B3
```

Durch diese Statements wird eine einzige Zeile gedruckt, die die Werte der folgenden Variablen enthält (gleichzeitig enthält die Tabelle die zu den Variablen gehörenden Formatangaben).

```
JL ——— I5
JM ——— I5
A1 ——— F18.6
A2 ——— F15.5
A3 ——— F15.6
B1 ——— F18.6
B2 ——— F15.6
B3 ——— F15.6
```

Wird dagegen die Anweisung

```
WRITE (6,3)  JL, JM, A1, A2, A3, B1, B2, B3, C1, C2, C3
```

geschrieben, so werden die Formatangaben von dem letzten geöffneten Klammerzeichen an wiederholt, d.h. von dem am weitesten rechts stehenden. Somit werden den zusätzlich in der WRITE-Anweisung auftretenden Variablen folgende Formatangaben zugeordnet:

```
C1 ←——— F18.6
C2 ←——— F15.6
C3 ←——— F15.6
```

Wenn Formatangaben auf diese Weise wiederholt werden, so wird grundsätzlich ein neuer Ausgabesatz begonnen. Wenn deshalb die Ausgabe über einen Drucker erfolgen soll, so muß man ein neues Steuerzeichen für den Drucker vorsehen. Damit würde aber für C1 die Formatangabe auf F17.6 reduziert werden, da die erste Stelle der Formatangabe F18.6 als diejenige interpretiert wird, die das Steuerzeichen enthält.

In der vollen FORTRAN-Sprache ist ein zusätzliches Klammernpaar innerhalb der FORMAT-Vereinbarungen erlaubt. Damit ist es möglich, auch Gruppen von Formatangaben zu wiederholen. Wenn es beim Programmablauf notwendig sein sollte, eine FORMAT-Vereinbarung zu wiederholen, so beginnt die Wiederholung bei der letzten geöffneten Klammer eines Klammernpaares der nullten oder ersten Ordnung. Die innersten Klammernpaare bleiben also unbeachtet, wenn drei Klammernpaare ineinander geschachtelt sind. Wir wollen diese Regel an einem Beispiel betrachten (Abb. 15.3.).

Um die einzelnen Klammernpaare bei diesem Beispiel besser verfolgen zu können, haben wir die einzelnen Klammernpaare durch ihre Ordnungsnummern gekennzeichnet. Wenn notwendig, so werden die Formatangaben von der letzten geöffneten Klammer erster Ordnung wiederholt. Diese Stelle haben wir durch den Pfeil gekennzeichnet.

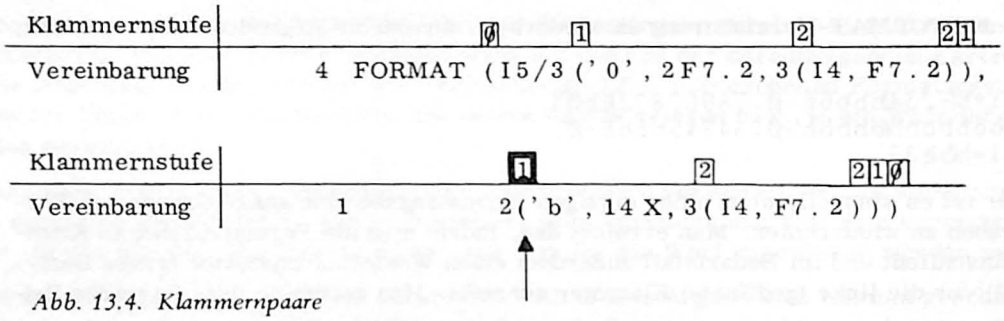


Abb. 13.4. Klammernpaare

Beim Umgang mit komplizierten Formaten sollten wir uns daran erinnern, daß der laufende Satz immer durch den ersten Schrägstrich oder durch das Ende der FORMAT-Vereinbarung beendet wird. Die Wiederholung eines Formats führt also immer zur Markierung des Anfanges eines neuen Satzes. Weiterhin weist jede Ausgabezeile, die irgendwann einmal fürs Drucken bestimmt ist, ihr eigenes Steuerzeichen auf.

Der Anfänger ist wahrscheinlich am besten beraten, wenn er mit FORMAT-Vereinbarungen beginnt, die relativ einfach aufgebaut sind; erst später sollte er sich mit Problemen beschäftigen, die ihn zwingen, sich mit komplizierten FORMAT-Vereinbarungen zu beschäftigen.

15.3. Weitere Formatangaben verschiedener Typen

a) Formatangaben vom T-Typ

Im allgemeinen werden FORMAT-Vereinbarungen benutzt, die den Aufbau des E/A-Satzes folgerichtig beschreiben, d.h. die Beschreibung beginnt bei der ersten Stelle des Satzes und schreitet bis zur letzten Stelle des Satzes fort. Die Formatangabe

Tw

bewirkt, daß die nachfolgenden Formatangaben für die Stellen des Satzes gelten, die mit der w-ten Stelle beginnen. Hierbei muß w < 256 sein. T-Formatangaben benutzt man hauptsächlich bei Überschriften; man betrachte dazu die folgenden Beispiele für die Druckerausgabe bzw. die Karteneingabe:

```
133 FORMAT   (T18,'UEBERSCHRIFT',F8.2,T6Ø,F8.4)
134 FORMAT   (T6,17,T1,I5)
```

Die FORMAT-Vereinbarung 133 (Druckerausgabe) druckt das Wort UEBERSCHRIFT auf die Druckstellen 17 bis 28 (man erinnere sich, daß das erste Zeichen eines Satzes bei Druckerausgabe als Steuerzeichen gilt und nicht gedruckt wird). Unmittelbar dahinter wird der Wert einer reellen Variablen in Form einer achtstelligen Zeichenfolge gedruckt; diese belegt die Druckstellen 29 bis 36. Bei der Druckstelle 59 beginnt dann die Ausgabe eines weiteren Wertes. Die restlichen Druckstellen der Zeile bleiben leer. Die FORMAT-Vereinbarung 134 (Karteneingabe) legt fest, daß von den Kartenspalten 6 bis 12 der Wert für eine Variable eingelesen wird, deren Wertevorrat ganzzahlig ist. Von den Spalten 1 bis 5 wird ebenfalls der Wert für eine solche Variable eingelesen. Eine READ-Anweisung, die auf die FORMAT-Vereinbarung 134 Bezug nimmt, könnte etwa folgendes Aussehen besitzen:

```
READ (5,134)  VAR2,VAR1
```

Der Wert, der VAR2 zugewiesen werden soll, wird von den Spalten 6 bis 12 gelesen, der Wert, der VAR1 zugewiesen werden soll, wird von den Spalten 1 bis 5 gelesen.

An Stelle der beiden eben besprochenen Statements für die Karteneingabe hätten wir auch die beiden Statements

```
135 FORMAT (I5,I7)
    READ (5,135)  VAR1,VAR2
```

verwenden können, weil diese uns zum gleichen Resultat geführt hätten. Die Formatangaben vom T-Typ, wie sie durch das Beispiel der FORMAT-Vereinbarung 133 demonstriert werden, ersparen dem Programmierer oft unnötige Überlegungen und Schreibarbeiten, wenn er sich mit der Zusammenstellung von Überschriftszeilen beschäftigen muß. Für andere Fälle ist die Verwendung von T-Formatangaben kaum diskutabel.

b) Skalenfaktor P

Für die D-, E- und F-Formatangaben kann ein Skalenfaktor verwendet werden; dieser muß vor der Formatangabe niedergeschrieben werden. Er besitzt folgende Form:

sP

Bei der Eingabe bleibt der Skalenfaktor unberücksichtigt, wenn die Eingabewerte die Buchstaben D bzw. E enthalten. Bei allen anderen Werten wird der Eingabewert mit 10^{-s} multipliziert, bevor er gespeichert wird. Bei der Ausgabe wird der im Hauptspeicher gespeicherte Wert zunächst mit 10^{s} multipliziert, bevor er ausgegeben wird. Bei D- und E-Formatangaben wird außerdem s vom Exponenten, d.h. von der Charakteristik, subtrahiert. Durch diese Subtraktion wird praktisch die Multiplikation wieder rückgängig gemacht, übrig bleibt deshalb nur eine Modifikation der normalen Ausgabe, d.h. der Dezimalpunkt ist um eine entsprechende Anzahl von Stellen verschoben. Eine derartige Korrektur erfolgt jedoch bei Werten, die unter Zugrundelegung von F-Formatangaben ausgegeben werden sollen, nicht.

Wird bei Formatangaben kein Skalenfaktor codiert, so wird zu Beginn der FORMAT Vereinbarung der Skalenfaktor ØP angenommen. Ist dagegen erst einmal ein Skalenfaktor zur Geltung gekommen, so bleibt dieser für den Rest der EA-Operation solange bestehen, bis ein neuer Skalenfaktor entdeckt wird. Ein einmal eingerichteter Skalenfaktor wird demnach auf alle nachfolgenden D-, E- und F-Formatangaben angewendet; er gilt jedoch nicht für I-Formatangaben. Skalenfaktoren können positiv oder negativ sein.

Nehmen wir einmal an, daß eine im Hauptspeicher gespeicherte Zahl mit 100 multipliziert werden muß, bevor sie ausgegeben werden kann (ist z.B. bei der Berechnung von Prozentsätzen der Fall). Man kann in einem solchen Fall auf die Codierung der Multiplikation verzichten und stattdessen einen geeigneten Skalenfaktor heranziehen; man würde beispielsweise einfach nur 2PF17.3 zu codieren brauchen. Unter Steuerung der Ausgabe durch diese Formatangabe findet die Multiplikation mit 10^{2} tatsächlich statt; der im Hauptspeicher stehende Wert wird dabei nicht verändert.

Wie schon gesagt, bedeutet die Ergänzung von D- bzw. E-Formatangaben durch einen Skalenfaktor, daß der Dezimalpunkt verschoben wird. Die Formatangabe E19.8 ergibt normalerweise bei der Ausgabe die Zeichenfolge

```
bbbb-Ø.12345678E+Ø9
```

[1]

Bei einem Skalenfaktor von 4, also bei 4PF19.8, würde sich folgendes Resultat ergeben:

```
B-1234.5678ØØØØE+Ø5
```

1) b bedeutet in diesem Abschnitt das Leerzeichen.

Bei einem Skalenfaktor von -4, also bei -4PF19.8, ergibt sich dagegen:

```
bbbb-Ø.ØØØØ1235E+13
```

Skalenfaktoren werden in erster Linie dazu verwendet, um mit F-Formatangaben eine Verschiebung des Dezimalpunktes nach links oder nach rechts zu erreichen, was einer Division durch bzw. einer Multiplikation mit einer Potenz von 10 gleichkommt. Bei Feldern, die unter Zugrundelegung der D- bzw. E-Formatangaben ausgegeben werden, erreicht man zwar durch Skalenfaktoren keine Wertänderung, doch kann man dafür sorgen, daß entweder vor dem Dezimalpunkt gültige Ziffern oder nach dem Dezimalpunkt zusätzliche Nullen erscheinen.

Will man die Wirksamkeit eines Skalenfaktors aufheben, so muß man vor die entsprechende Formatangabe ØP schreiben.

c) Formatangaben vom Z-Typ

Die Z-Formatangaben besitzen folgende allgemeine Form:

Zw

Hierbei wird durch w die Feldlänge spezifiziert. Bei dieser Formatangabe wird angenommen, daß das so festgelegte Feld auf dem Datenträger sedezimale Ziffern enthält. Führende Leerzeichen und Leerzeichen am Schluß werden als die Sedezimalziffer Ø angesehen. Bei der Eingabe wird das externe Feld rechtsbündig der Variablen zugewiesen, jede Konvertierung, sieht man einmal von der Ersetzung der Leerzeichen durch Nullen ab, unterbleibt dabei. Ist das für die Variable im Hauptspeicher zur Verfügung stehende Feld zu klein, so gehen die führenden Sedezimalziffern verloren. Bei der Ausgabe werden den externen Feldern je nach Bedarf entweder führende Leerzeichen hinzugefügt oder die führenden Sedezimalziffern des internen Wertes werden abgeschnitten. Betrachten wir ein Beispiel.

```
188 FORMAT (Z9,I5)
    READ (5,188)  J,K
```

Auf der Eingabekarte möge die Ziffernfolge

```
234598ACB-Ø123
```

vorhanden sein. Nach der Eingabe besitzen J und K Werte, die, sedezimal gesehen, folgendes Aussehen haben:

J	34 59 8A CB
K	ØØ ØØ ØØ 7B

d) Formatangaben vom G-Typ

Diese Formatangaben stehen nur in der vollen Sprache zur Verfügung. Sie besitzen folgende allgemeine Form:

Gw.s oder
Gw

Hierbei gibt w die Feldlänge und s die Anzahl der signifikanten Ziffern an. Man beachte, daß bei der G-Formatangabe die Anzahl der signifikanten Ziffern (d.h. die Ziffernfolge der Zahl ohne führende Nullen) erscheint und nicht die Anzahl der Dezimalstellen. Die Formatangabe G kann anstelle von D-, E-, F- und I-Formatangaben verwendet werden.

Bei ganzzahligen Variablen bleibt s unberücksichtigt, falls es niedergeschrieben sein sollte und die Formatangabe Gw wird als gleichwertig der Formatangabe Iw betrachtet.

Bei der Eingabe von Werten für reelle Variablen wird die G-Formatangabe genau so wie die F-Formatangabe behandelt.

Für die Ausgabe von Werten reeller Variablen müssen wir einige Beziehungen kennenlernen. Sei N der Wert einer reellen Variablen, so gilt

I. $N < 0,1$
Die Ausgabe erfolgt unter Zugrundelegung der Formatangabe Ew.d mit einem Skalenfaktor

II. $10^{k-1} \leqq N \leqq 10^k$ für $k=0,1,2,\cdots,s$
Die Ausgabe erfolgt hier nach der F-Formatangabe mit einer Feldbreite von (w-4) Stellen, wobei (s-k) Stellen Dezimalstellen sind; an dieses Feld schließen sich 4 Leerstellen an. Zusammengefaßt, die Ausgabe erfolgt also nach F(w-4).(s-k),4X

III. $10^s \leqq N$
Die Ausgabe erfolgt nach der Formatangabe Ew.d mit einem Skalenfaktor.

Liegt N zwischen 0,1 und 10^s, so wird also die Zahl unter Zugrundelegung einer F-Formatangabe ausgegeben; der F-Formatangabe folgt eine X-Formatangabe. Ein Skalenfaktor geht hierbei in die Ausgabe nicht ein. In allen anderen Wertebereichen wird die Zahl nach der E-Formatangabe ausgegeben, wobei der maßgebende Skalenfaktor benutzt wird. Ein paar Beispiele mögen diese Sachlage verdeutlichen. Nehmen wir also an, daß die Formatangabe G18.8 vorliegt und auf die Variablen angewendet wird, die in der Abb. 15.4. aufgeführt sind.

J = 7	bbbbbbbbbbbbbbbb7
M = -81345	bbbbbbbbbbbb-81345
X = Ø.ØØØ72	bbbbØ.72ØØØØØØE-Ø3
Y = Ø.72	bbbbØ.72ØØØØØØbbbb
Z = 72Ø.Ø	bbbbb72Ø.ØØØØØbbbb
T = 72ØØØØØ.Ø	bbbbb72ØØØØØ.Øbbbb
U = 72ØØØØØØ.Ø	bbbbb72ØØØØØØ.bbbb
V = 72ØØØØØØØ.Ø	bbbbØ.72ØØØØØØE+Ø9

Anmerkung: b bedeutet hier das Leerzeichen

Abb. 15.4. Ausgabe im G-Format

Aufgabengruppe 39

1) Auf einer Karte sind A, B und C im Format F18.7 abgelocht. Schreibe ein Programm, das diese Karten einliest und zunächst folgende Zeile ausdruckt:

A=···, B=···, C=···,

Damit werden also die Eingabewerte fixiert. Anschließend sind die reellen Lösungen der Gleichung

$Ax^2 + Bx + C = 0$

zu bestimmen. Aufgrund der Rechenergebnisse ist danach entweder die Zeile

DIE GLEICHUNG HAT KEINE REELLEN LOESUNGEN

oder die Zeile

LOESUNGEN: X1=··· UND X2=···

zu drucken. Nach dem Druck dieser Zeile ist eine Zeile freizulassen, und die Programmschritte sind für die nachfolgende Datenkarte zu wiederholen.

2) Ein Programm zur Bestimmung des Tilgungsplanes von Hypotheken soll dem Maschinenbediener die Möglichkeit bieten, Karten einzugeben, auf denen Hypothekenhöhe, jährlicher Zinsfuß und monatliche Ratenzahlung abgelocht sidn. Das Programm soll geeignete Überschriften und den Zahlungspaln ausdrucken. Die Ausgabeliste könnte z.B. folgendes Aussehen haben:

```
HYPOTHEK = 1ØØØØ.ØØ DM
ZINSFUSS = 6.ØØ %
MONATSRATE = 2ØØ.ØØ DM

MONAT        RATE        ZINSEN      TILGUNG     KONTENSTAND
                                                 1ØØØØ.ØØ
  1         2ØØ.ØØ       5Ø.ØØ      15Ø.ØØ       985Ø.ØØ
  2         2ØØ.ØØ       49.25      15Ø.75       9699.25
  :
  :
```

3) Ändere die Ausgabe von Aufgabe 2) so ab, daß innerhalb der Tabelle nach jeder 5. Zeile eine Leerzeile verbleibt.

4) Ein Kartenpaket enthält drei verschiedene Kartenarten. Alle Karten sind wie folgt abgelocht:

Spalten 1 bis 4	:	Kontonummer (alphanumerisch), z.B. KL87
Spalte 10	:	Kartenart
Spalten 11 bis 20:		Betrag (Format F10.2)

Karten der Kartenart 1 enthalten den alten Kontenstand, Karten der Kartenart 2 sind für Einzahlungen, Karten der Kartenart 3 für Auszahlungen vorgesehen. Alle Karten sind nach aufsteigenden Kontonummern sortiert; innerhalb gleicher Kontonummern können jedoch die Karten in beliebiger Reihenfolge erscheinen. Für jede Kontonummer liegt eine Karte der Kartenart 1 vor, Karten der Kartenarten 2 und 3 können in beliebiger Anzahl nachfolgen, sie müssen jedoch nicht unbedingt vorhanden sein. Im Programm soll der neue Kontenstand eines jeden Kontos errechnet werden und in eine Karte der Kartenart 1 zusammen mit der Kontonummer und der Kartenart gestanzt werden.

5) Die Aufgabe 4) ist so zu ergänzen, daß zusätzlich die Nachricht

KONTO xxxx KONTENSTAND -xxxxxx.xx

gedruckt wird, wenn der neue Kontenstand eines Kontos ein Soll aufweist.

6) Aufgabe 5) ist so zu modifizieren, daß für Konten mit einem Habensaldo von über 25000.- DM ebenfalls eine Zeile gedruckt wird. Der Kontenstand soll hierbei nach rechts versetzt erscheinen, d.h. nicht unter die Kontenstände mit negativen Beträgen gedruckt werden.

7) Aufgabe 4) ist so abzuändern, daß bei negativen Kontenständen das Minuszeichen nicht vor die Zahl gestanzt wird, sondern auf die Spalte 21.

8) Schreibe ein Programm, das Karten der Kartenart 1 (siehe Aufgabe 4) einliest und Karten der Kartenart 4 ausstanzt, die die gleiche Einteilung wie die anderen Karten besitzen. Auf der Spalte 10 soll jedoch eine 4 und auf die Spalten 11 bis 20 der Zinsbetrag bestanzt werden. Die Zinsen sollen aus dem laufenden Zinssatz p errechnet werden, indem dieser zunächst durch 12 dividiert und anschließend mit dem Betrag aus den Spalten 11 bis 20 der eingelesenen Karten multipliziert wird. Vor die Karten der Kartenart 1 wird eine Vorlaufkarte gelegt; diese enthält eine 0 auf der Spalte 10 und den gegenwärtigen Zinssatz auf den Spalten 11 bis 20. Der Dezimalpunkt wird beim Zinssatz grundsätzlich mit abgelocht.

15.4. Indizes

In einem FORTRAN-Programm dürfen die Variablen indiziert werden. Die Indizes können als Ausdruck [1] angegeben werden, vorausgesetzt, der Ausdruck entspricht den Regeln der FORTRAN-Sprache. Bei der Programmausführung wird der Ausdruck berechnet und in eine ganze Zahl umgewandelt. Der sich so ergebende Wert muß positiv sein. Indizes werden in Klammern eingeschlossen und an den Variablennamen angehängt, zu dem sie gehören. Mehrere Indizes werden untereinander durch Kommas getrennt. Bis zu sieben verschiedene Indizes [1] sind erlaubt. Einige Beispiele mögen unsere Betrachtungen untermauern:

```
X(7)
Y(KAPPA,BULLE)
Z(GELD-5,KRUMM+3*TOUR)
MARI(ANNE+2,J,K,L,4**J,KARL,L-2)
```

Für jede indizierte Variable muß der maximale Umfang des Bereiches, d.h. die maximalen Werte, die die Indizes annehmen dürfen, einmal an einem beliebigen Platz im Programm festgelegt werden, damit der Kompilierer genügend Hauptspeicherplatz für das Objektprogramm reservieren kann. Die hierzu erforderlichen Dimensionsfestlegungen erfolgen meist in der DIMENSION-Vereinbarung, die in der Form

```
DIMENSION v1,v2,...
```

niederzuschreiben ist. Hierbei bedeuten die v_i die Bereichsvariablen. Die Vereinbarung

```
DIMENSION  A(10),B(5,3),K(4),D(2,2,3)
```

bewirkt z.B., daß für A Speicherplatz für 10 Werte, für B Speicherplatz für $5 \cdot 3 = 15$ Werte, für K Speicherplatz für 4 Werte und für D Speicherplatz für $2 \cdot 2 \cdot 3 = 12$ Werte freigehalten wird. In einer einzigen DIMENSION-Vereinbarung dürfen beliebig viele Variablen dimensioniert werden; in einem Quellenprogramm dürfen so viele DIMENSION-Statements verwendet werden, wie man nur immer für wünschenswert erachtet.

Es ist jedoch notwendig, daß die Dimensionen für jede indizierte Variable vor Verwendung derselben in anderen Statements erklärt werden.

DIMENSION-Statements sind nichtausführbare Statements (Vereinbarungen), d.h. aus ihnen werden keine Instruktionen für das Objektprogramm generiert. Sie bleiben also beim Übergang von einem Statement zu einem anderen Statement (Programmablaufsteuerung) gänzlich unbeachtet.

---

1) Hier liegen beachtliche Spracherweiterungen vor, die wesentlich über die Normierung der FORTRAN-Sprache hinausgehen.

Besitzt eine Bereichsvariable nur eine Dimension, so werden die Werte in aufsteigender Reihenfolge der Indizes gespeichert, z.B. K(1), K(2), K(3) usw. Bei zwei Dimensionen werden die einzelnen Bereichselemente spaltenweise gespeichert, d.h. der erste Index ändert sich am häufigsten. [1)]

Betrachten wir z.B. die Matrix

$$\begin{bmatrix} B_{11} & B_{12} & B_{13} & B_{14} & B_{15} \\ B_{21} & B_{22} & B_{23} & B_{24} & B_{25} \\ B_{31} & B_{32} & B_{33} & B_{34} & B_{35} \end{bmatrix}$$

Die Elemente dieser Matrix werden in der folgenden Reihenfolge gespeichert:

B(1, 1), B(2, 1), B(3, 1), B(1, 2), B(2, 2), B(3, 2), B(1, 3), ···, B(2, 5), B(3, 5).

Dasselbe Prinzip wird auch auf Bereiche angewendet, die mehr als zwei Dimensionen aufweisen: Der erste Index variiert am häufigsten, der nächste Index am zweithäufigsten, der am weitesten rechts stehende ändert sich am wenigsten. Für eine Variable mit drei Dimensionen käme es deshalb zu folgender Speicherungsreihenfolge der Elemente:

D(1, 1, 1), D(2, 1, 1), D(1, 2, 1), D(2, 2, 1), D(1, 1, 2), ···, D(1, 2, 3), D(2, 2, 3).

Mit Ausnahme einiger noch zu besprechender Fälle und in Datenlisten bei Eingabe/Ausgabe-Anweisungen ist es sonst nicht erlaubt, sich auf eine indizierte Variable ohne Angabe aller Indizes zu beziehen.

Bei der vollen Sprache ist es erlaubt, die Informationen über die Dimensionen einer Variablen in die expliziten Typvereinbarungen aufzunehmen, d.h. in die Statements REAL, INTEGER usw. Dabei wird der Maximalwert eines Index genau so niedergeschrieben, wie er in der DIMENSION-Vereinbarung erscheinen würde. Beispiele:

```
REAL*8   YR(12),B*4(5,3)
INTEGER*2  MKR(3),MKS(4Ø2),W*4(48)
```

Es ist jedoch immer erlaubt, sie auch in der Form anzugeben, die für die BASIC FORTRAN-Sprache verbindlich ist. Wir können somit auch codieren:

```
DOUBLE PRECISION  YR
DIMENSION  YR(12),B(5,3)
```

Damit können wir das Statement

```
REAL*8   YR(12),B*4(5,3)
```

ersetzen.

15.5. DO-Anweisungen

Die komplizierteste, doch zugleich auch die schwerwiegendste Steueranweisung ist die DO-Anweisung. Ihre allgemeine Form lautet:

DO n $i=m_1,m_2,m_3$

1) Im Gegensatz zu PL/1: Dort ändert sich der letzte Index am häufigsten, d.h. die Elemente werden zeilenweise gespeichert.

Statt dieser Form kann sie auch in der etwas einfacheren Form

```
DO n    i=m1,m2
```

angewendet werden. Bei beiden Formen haben die einzelnen Größen folgende Bedeutung:

n	ist eine Statementnummer,
i	ist eine nichtindizierte Variable, die Lautvariable genannt wird,
m_1, m_2, m_3	sind ganzzahlige Konstanten ohne Vorzeichen oder auch nichtindizierte Variablen des ganzzahligen Typs, die zur Ausführungszeit der DO-Anweisung positive ganzzahlige Werte besitzen müssen. Fehlt die Angabe von m_3, so wird es zu 1 angenommen.

Man betrachte z.B. die folgenden beiden Anweisungen:

```
DO 3    I=2,1Ø,2
DO 4    J=1,K
```

Durch das Schlüsselwort DO wird eine Anweisung festgelegt, die die wiederholte Ausführung der ihr folgenden Statements bis einschließlich des durch die Statementnummer n markierten Statements zur Folge hat. Beim ersten Durchlauf durch die somit gebildete Schleife besitzt die Laufvariable i den Wert m_1, beim zweiten Durchlauf den Wert m_1+m_3; der letzte Schleifendurchlauf ergibt sich, wenn die Laufvariable i den größten Wert angenommen hat, der gleich oder kleiner als m_2 ist.

Ist $m_1>m_2$, so wird die Schleife nur ein einziges Mal durchlaufen.

Unter der Schleifenspanne einer DO-Anweisung wollen wir alle Statements verstehen, die wiederholt ausgeführt werden. Nach der letzten Ausführung des Schleifenbereiches spricht man davon, daß die DO-Anweisung erfüllt ist. Danach geht die Programmablaufsteuerung zu der Anweisung über, die auf die Anweisung folgt, die durch n markiert ist. Man sagt in diesem Fall, daß die DO-Schleife [1] durch den Normalausgang verlassen wird.

Während der einzelnen Schleifendurchläufe stehen die Werte der Laufvariablen i für Berechnungen in genau derselben Weise zur Verfügung wie die Werte jeder anderen Variablen des ganzzahligen Typs. Wenn die Programmablaufsteuerung durch irgendeine Steueranweisung (z.B. GO TO oder IF) zu einem Statement außerhalb der Schleifenspanne führt, so bleibt der Wert der Laufvariablen ebenfalls erhalten. Wird jedoch die Schleifenspanne normal verlassen, so wird die Laufvariable zu einer undefinierten Größe, d.h. sie besitzt keinen Wert mehr. Auf die Laufvariable i kann man sich infolgedessen erst dann wieder beziehen, wenn sie neu definiert wird, d.h. wenn sie entweder auf der linken Seite einer arithmetischen Ergibtanweisung oder in der Datenliste einer Eingabeanweisung (READ) oder in einer neuen DO-Anweisung als Laufvariable erscheint.

Am Beispiel in Abb. 15.5. soll das ganze noch einmal veranschaulicht werden:

Die Variablen T nimmt beim Duchlaufen der Schleifenspanne der Reihe nach die Werte 6.5 — 5.5, 7.5 — 5.5, 7.5 — 4.5, 6.5 — 2.5, 4.5 — -0.5 an. Zu der Zeit, zu der sie den Wert -0.5 erreicht hat, verzweigt das Programm zu der Anweisung, die durch die Statementnummer 8 markiert ist. Die Laufvariable J besitzt in diesem

1) DO-Schleifen sind „induktive" Schleifen, da ein in die Schleifenspanne eingeführter Zähler (die Laufvariable nämlich) die Anzahl der Schleifendurchläufe kontrolliert.

```
      T = 6.5
      DO 3  J=1,7
         TJ = J
         T = T - TJ
         IF  (T)  8,3,3
3     T = T + 2.Ø
8  XZW = T*T
```

Abb. 15.5. Beispiel für eine induktive Schleife

Augenblick den Wert 5. Beginnen wir jedoch unser Programmbeispiel mit dem Anfangswert 26.5 für T, so nimmt T bei Ausführung der Anweisung 3 der Reihe nach die folgenden Werte an: 27.5, 27.5, 26.5, 24.5, 21.5, 17.5, 12.5. Ist aber T=12.5, so wird die Schleifenspanne normal verlassen, da J den Wert 7 angenommen hat. Da bei einem Normalausgang die Laufvariable zu einer undefinierten Größe wird, steht J für Weiterrechnungen nicht mehr zur Verfügung. Will oder muß man nun mit dem zuletzt erreichten Wert von J weiterarbeiten, so bleibt dem Programmierer nichts anderes übrig, als unmittelbar vor der Anweisung mit der Statementnummer 8 eine zusätzliche Anweisung, nämlich

```
J=7
```

einzuschieben.

Die nachfolgend aufgeführten Regeln gelten für die Schleifenspanne von DO-Anweisungen:

1) Regel 1
   Das erste Statement einer Schleifenspanne muß eine Anweisung sein. Eine Vereinbarung, wie REAL, INTEGER, DIMENSION usw. darf nicht an erster Stelle innerhalb eines solchen Bereiches stehen.

2) Regel 2
   Das letzte Statement einer Schleifenspanne darf weder eine GO TO-Anweisung noch eine arithmetische IF-Anweisung sein. Um diese Regel zu umgehen, muß das Statement CONTINUE codiert werden. Somit ist die Statementfolge

```
1  DO 5  J=1,25,2
      IF  (X-Y(J))  5,4,4
5  CONTINUE
```

   im Sinne dieser Regel eine gültige Statementfolge.

   Das Statement CONTINUE ist eine Vereinbarung, denn bei der Übersetzung des Quellenprogrammes werden aus ihm keine Instruktionen für das Objektprogramm erzeugt. Seine einzige Funktion besteht also darin, daß es als letztes Statement der Schleifenspanne einer DO-Anweisung eingesetzt wird. Auf diese Weise kann man den eindeutigen Abschluß einer solchen Spanne festlegen, wenn dieser sonst durch eine Verzweigungsanweisung abgeschlossen werden müßte.

3) Regel 3
   In einer Schleifenspanne darf keine Anweisung aufgenommen werden, die den Wert der Laufvariablen selbst oder deren Anfangswert oder deren Endwert oder deren Inkrement (Erhöhungswert) verändert. Mit anderen Worten: Anfangswert, Endwert und Erhöhungswert der die Schleife steuernden Laufvariablen müssen vor dem Eintritt in die Schleife bereits unwiderruflich festgelegt sein.

4) Regel 4

In der Schleifenspanne einer DO-Anweisung dürfen weitere DO-Anweisungen auftreten. In solchen Fällen müssen jedoch die Schleifenspanne der inneren DO-Anweisungen stets voll von den Schleifenspannen aller äußeren DO-Anweisungen umschlossen sein. Für die Anzahl solcher ineinandergeschachtelter DO-Anweisungen bestehen keinerlei Beschränkungen. Die Menge der DO-Anweisungen, die dieser Regel genügt, wird oft auch als DO-Nest oder als DO-Schachtelung bezeichnet. Betrachten wir als Beispiel das Nest in Abb. 15.6.

```
      DIMENSION  A(5Ø,6)
1Ø    DO 7   N=1,5Ø
      DO 7   I=1,6
      XK = I
 7    A(N,I) = (1.Ø+Ø.Ø1*XK) ** N
```

Abb. 15.6. Ineinandergeschachtelte induktive Schleifen

Dieser Programmabschnitt berechnet

$$a_{n,i} = (1+i)^n \quad \text{für} \quad i=Ø.Ø1, Ø.Ø2, \cdots, Ø.Ø6 \quad \text{und für } n=1,2,\cdots,5Ø$$

Man beachte hierbei, daß die Schleifenspanne zweier ineinandergeschachtelter DO-Anweisungen durchaus das gleiche letzte Statement besitzen können; sie müssen jedoch nicht das gleiche letzte Statement haben.

5) Regel 5

Es ist keine Verzweigung durch eine GO TO-Anweisung oder durch eine IF-Anweisung in die Schleifenspanne einer DO-Anweisung erlaubt, wenn die Verzweigungsstelle außerhalb der Schleifenspanne liegt. (Eine solche Verzweigung verhindert das korrekte Aufsetzen des Anfangswertes der Laufvariablen). Diese Regel verbietet somit auch eine Verzweigung von der Schleifenspanne einer äußeren DO-Anweisung in die Schleifenspanne einer inneren DO-Anweisung. Eine Verzweigung in der umgekehrten Richtung, also von der Schleifenspanne einer inneren DO-Anweisung in die Schleifenspanne einer äußeren DO-Anweisung, ist dagegen erlaubt, da eine solche Verzweigung vom Standpunkt der äußeren DO-Anweisung nur von einer Stelle innerhalb ihrer Schleifenspanne zu einer anderen Stelle ihrer Schleifenspanne führt.

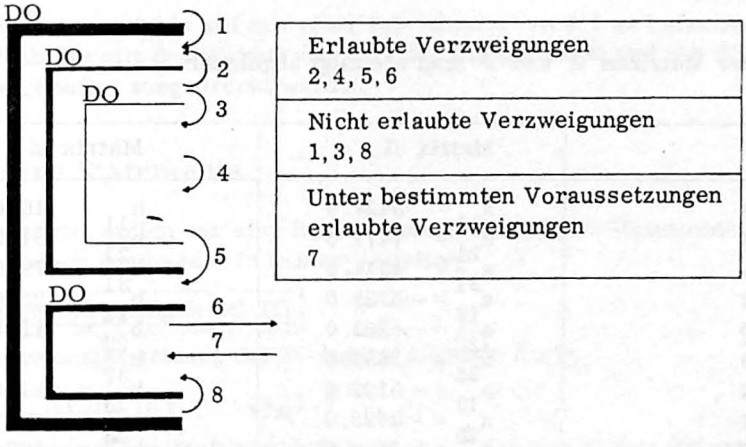


Abb. 15.7. DO-Schachtelungen

Diese Regel läßt eine Ausnahme zu: Es ist erlaubt, die Schleifenspanne einer DO-Anweisung zu verlassen und sie später wieder zu betreten; es darf in der Zwischenzeit allerdings keine Wertänderung der Laufvariablen oder des Anfangswertes oder des Endwertes oder des Inkrementes derselben vorgenommen werden. Diese Möglichkeit läßt sich z.B. wunderbar zur Ausführung von Unterprogrammen einsetzen. Erlaubte und nicht erlaubte Verzweigungen sind in der Abb. 15.7. anschaulich dargestellt.

Aufgabengruppe 40

1) Schreibe ein Programm, mit dessen Hilfe die kleinste Zahl unter den 50 Zahlen a_1, a_2, a_3, ···, a_{50} bestimmt werden kann. Die kleinste Zahl ist der Variablen WINZI als Wert zuzuweisen.

2) Schreibe ein Programm, das die natürlichen Zahlen von 1 bis 100 generiert und sie als Werte den Bereichselementen J(I) zuweist. Es gelte dabei, daß j_i=i ist.

3) Ergänze das Programm zu Aufgabe 2) so, daß außerdem die Quadrate der natürlichen Zahlen in Gleitpunktdarstellung errechnet werden und den Elementen des Bereichs XQ als Werte zugewiesen werden.

4) Zu welchem Resultat hinsichtlich der 50 Elemente des Bereiches A (siehe Aufgabe 1) führt das folgende Programm?

```
      DIMENSION  A(5Ø),B(5Ø)
      N = 5Ø
      DO 12  I=1,N
12       B(I) = A(I)
      M = N - 1
      DO 13  I=1,M
         K = I + 1
         DO 13  J=K,N
            IF  (B(J)-B(I))  14,13,13
13    CONTINUE
      STOP
14    C = B(J)
      B(J) = B(I)
      B(I) = C
      GO TO 13
      END
```

5) Die Elemente der Matrizen $\mathcal{A}$ und $\mathcal{B}$ sind wie folgt abgelocht:

Spalten	Matrix $\mathcal{A}$	Matrix $\mathcal{B}$
1 bis 8	a_{11} = 3124.0	b_{11} = 1616.0
9 bis 16	a_{21} = 7412.0	b_{21} = 8182.0
17 bis 24	a_{31} = 1314.0	b_{31} = 7919.0
25 bis 32	a_{12} = - 5329.0	b_{12} = 2516.0
33 bis 40	a_{22} = - 4203.0	b_{22} = 1228.0
41 bis 48	a_{32} = - 2031.0	b_{32} = - 8243.0
49 bis 56	a_{13} = - 9122.0	b_{13} = - 2432.0
57 bis 64	a_{23} = - 3425.0	b_{23} = 4325.0
65 bis 72	a_{33} = 1444.0	b_{33} = 1334.0

Alle Kartenfelder sind im Format F8.Ø abgelocht. Die Matrix $\mathcal{L}$ bestehe aus den Elementen

$$c_{ij} = a_{ij} + b_{ij}$$

Sie heißt deshalb die Summenmatrix der beiden Matrizen $\mathcal{A}$ und $\mathcal{B}$.

Schreibe ein Programm in der FORTRAN-Sprache, das die zwei Karten mit den Elementen von $\mathcal{A}$ und $\mathcal{B}$ einliest und das auf eine Karte die Elemente der Summenmatrix $\mathcal{L}$ in der gleichen Reihenfolge stanzt!

6) Die Matrix $\mathcal{D}$, die aus den Elementen

$$d_{ij} = \sum_{k=1}^{3} a_{ik} \quad b_{kj}$$

besteht, heißt Produktmatrix der beiden Matrizen $\mathcal{A}$ und $\mathcal{B}$.
Schreibe ein Programm, das die Elemente von $\mathcal{A}$ und $\mathcal{B}$ einliest und die Elemente von $\mathcal{D}$ stanzt (Ein- bzw. Ausgabe wie bei Aufgabe 5)!

7) Schreibe ein Programm, das die Elemente einer quadratischen Matrix mit drei Zeilen und drei Spalten einliest und die Determinante der Matrix errechnet und ausgibt.

8) Schreibe ein Programm, das die Elemente einer Matrix 3. Ordnung einliest und in folgender Anordnung ausdruckt:

$$\begin{array}{lll} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{23} & a_{33} \end{array}$$

Die Matrixelemente sollen also genau so ausgedruckt werden, wie man eine Matrix gewöhnlich schreibt.

9) In einem Programm ist

$$S = \sum_{n=1}^{\infty} \frac{x}{n(n+1)}$$

für das Intervall von x=0 bis x=1 mit einer Schrittweite von 0,1 zu berechnen.
Es soll eine Tabelle mit den Werten von x (in der linken Spalte) und den Werten von S (in der rechten Spalte) ausgedruckt werden.

15.6. Weitere Statements

In diesem Abschnitt wollen wir eine Reihe weiterer FORTRAN-Statements kennenlernen. Diese stehen jedoch zueinander in keiner Beziehung.

1) Rechensprunganweisungen GO TO

Die Rechensprunganweisung GO TO besitzt folgende Form:

```
GO TO (n1,n2,n3,···,nm),i
```

Hierbei bedeuten die n_i Statementnummern und i ist eine nichtindizierte Variable des ganzzahligen Typs. Wenn zur Ausführungszeit des Statements die Variable i den Wert

j besitzt und gleichzeitig die Beziehung $1 \leqq j \leqq m$ erfüllt ist, so erfolgt eine Verzweigung zum Statement mit der Nummer n_j. Somit bewirkt die Anweisung

```
GO TO (3,7,6,6),K
```

einen Sprung zum Statement 3, wenn K gleich 1 ist, einen Sprung zum Statement 7, wenn K gleich 2 ist usw. Wenn eine Rechensprunganweisung ausgeführt werden soll und der Wert von i liegt nicht im Wertebereich von 1 bis einschließlich m, so kommt das nächstfolgende Statement zum Tragen.

2) EQUIVALENCE

Zur Einsparung von Speicherplatz im Objektprogramm kann das folgende Statement eingesetzt werden:

```
EQUIVALENCE (a,b,···), (d,e,f,···) ···
```

In dieser Vereinbarung stellen a, b, c, ··· die Namen von nichtindizierten Variablen oder von Bereichselementen mit konstantem Index dar. Verfolgen wir ein Beispiel:

```
EQUIVALENCE (A,F,G), (J,K,I(3))
```

Durch diese Vereinbarung erreichen wir, daß den Variablen A, F und G die gleiche Speicheradresse zugeordnet wird. Dasselbe gilt auch für die Variablen J und K sowie für das Bereichselement I(3), denen ebenfalls ein gleicher Platz, der jedoch ein anderer als bei den ersten drei Variablen ist, zugeordnet wird. Somit benötigen wir nur zwei Felder an Stelle von ursprünglich sechs. Variablen, die dem gleichen Speicherplatz zugeordnet werden sollen, müssen jedoch die gleiche Länge aufweisen.

Besitzt eine Variable mehr als einen Index, so ist es erlaubt, diese in die EQUIVALENCE-Vereinbarung entweder mit ihren sämtlichen Indizes oder nur mit einem einzigen Index aufzunehmen. Im letzteren Fall bezieht sich der Index auf die Stellung des betreffenden Bereichselementes innerhalb des Bereiches, dessen Elemente ja bekanntlich spaltenweise im Hauptspeicher aufeinander folgen. Wenn wir z. B. die Statementfolge

```
DIMENSION  EX(7,3),EY(4)
EQUIVALENCE  (EX(2),RT)
```

betrachten, so wird dem Bereichselement EX(2, 1) der gleiche Speicherplatz wie der Variablen RT zugeordnet. Das Statement

```
EQUIVALENCE  (EX(8),RU)
```

ordnet in gleicher Weise EX(1, 2) und RU den gleichen Speicherplatz zu. Die Vereinbarung

```
EQUIVALENCE  (EX(1,2),EY(2))
```

bewirkt, daß die beiden Bereichselemente EX(1, 2) und EY(2) gemeinsam den gleichen Speicherplatz benutzen. Damit ist aber eingeschlossen, daß sich auch andere Elemente von EX und EY mit einem gemeinsamen Speicherplatz begnügen müssen. Es kommt zu der Übereinstimmung von Bereichselementen, die in Abb. 15.8. dargestellt ist.

Am meisten wird wohl die EQUIVALENCE-Vereinbarung dann verwendet, wenn es darum geht, Irrtümer zu korrigieren, die daraus entstanden sind, daß man sich auf eine Variable mit zwei oder mehr verschiedenen Namen bezogen hat.

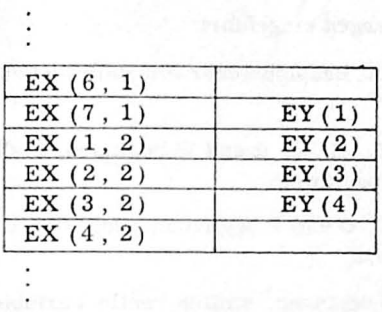


Abb. 15.8. Übereinstimmung von Bereichselementen

Die beiden folgenden Statements stehen nur in der vollen FORTRAN-Sprache zur Verfügung. Die Erfahrung hat gezeigt, daß sie kaum sehr vorteilhaft angewendet werden können.

3) ASSIGN

Die Anweisung

```
ASSIGN i TO m
```

bewirkt, daß der nichtindizierten Variablen m der Wert i zugewiesen wird; i muß hierbei eine Statementnummer sein. Eine auf diese Weise verwendete Variable m kann nur in einem anderen ASSIGN-Statement oder in einer Anweisung für einen gesetzen Sprung verwendet werden. Obwohl i der Variablen m als Wert zugewiesen wird, ist der im Hauptspeicher gespeicherte Wert von m nicht gleich i, sondern vielmehr eine Adresse.

4) Anweisungen für gesetzte Sprünge

Die allgemeine Form dieser Anweisungen sieht wie folgt aus:

```
GO TO m,(x1,x2,···,xn)
```

Unter den x_i sind Statementnummern zu verstehen. Die Programmablaufsteuerung geht zu dem Statement über, dessen Nummer der Variablen m durch ein vorausgehendes ASSIGN-Statement zugewiesen wurde. Der Wert von m muß gleich einer der in Klammern stehenden Statementnummern x_i sein.

5) IMPLICIT

Die expliziten Typvereinbarungen REAL, INTEGER und DOUBLE PRECISION können dazu verwendet werden, um den Typ und die Länge der im einzelnen aufgeführten Variablen festzulegen. Die Vereinbarung IMPLICIT wird dazu benutzt, Typ und Länge für alle Variablen festzulegen, deren Namen mit einem bestimmten Buchstaben des Alphabets oder mit einem Buchstaben aus einer bestimmten Buchstabengruppe beginnen. Diese Vereinbarung besitzt folgende allgemeine Form:

```
IMPLICIT typ*s(a1,···,an), ···, typ*s(···)
```

Durch die a_i werden entweder ein Buchstaben (z.B. A, G usw.) oder zwei, durch einen Bindestrich getrennte Buchstaben (z.B. A-F), spezifiziert. An Stelle von typ*s kann REAL*8, REAL*4, INTEGER*4 oder INTEGER*2 geschrieben werden. Wird s weggelassen, so wird REAL*4 bzw. INTEGER*4 angenommen. Wir wollen uns ein Beispiel für diese Vereinbarung ansehen:

```
IMPLICIT INTEGER*(A), INTEGER*4(I-P), REAL*8(B-H,Q-R,W), REAL(S-V,$)
```

Durch diese Vereinbarung werden folgende Typfestlegungen eingeführt:

a) Alle Variablen, deren Namen mit A beginnt, wird im Hauptspeicher ein Halbwort zugeordnet; außerdem sind sie vom Typ ganzzahlig,

b) Alle Variablen, deren Namen mit B, C, D, E, F, G, H, Q, R und W beginnen, stellen reelle Variablen dar und belegen im Hauptspeicher ein Doppelwort,

c) Alle Variablen, deren Namen mit I, J, K, L, M, N, O und P beginnen, stellen ganzzahlige Variablen dar und belegen im Hauptspeicher ein Wort,

d) Alle Variablen, deren Namen mit S, T, U, V und $ beginnen, stellen reelle Variablen dar und belegen im Hauptspeicher ein Wort.

Es wäre noch zu bemerken, daß die Variablen, deren Namen mit den Buchstaben X, Y und Z beginnen, durch diese Vereinbarung hinsichtlich des Typs nicht festgelegt werden, so daß für sie der ursprüngliche Charakter erhalten bleibt, d.h. hier REAL*4.

Der durch die IMPLICIT-Vereinbarung festgelegte Typ kann selbstverständlich durch eine nachfolgende explizite Typvereinbarung zunichte gemacht werden. Bleiben wir bei unserem Beispiel: Durch

```
INTEGER GYI
```

schalten wir die o.a. IMPLICIT-Vereinbarung für die angegebene Variable aus, so daß GYI als ganzzahlige Variable mit der Länge 4 angesehen wird.

6) CALL DUMP

Dieses Statement besitzt die Form

CALL DUMP(a_1,b_1,f_1,···,a_n,b_n,f_n)

Stößt die Ablaufsteuerung eines Programmes bei der Ausführung desselben auf ein Statement dieser Form, so wird die Ausführung des Objektprogrammes eingestellt und der Inhalt der Hauptspeicherbereiche zwischen a_1 und b_1 bzw. a_2 und b_2 usw. ausgegeben. In diesem Hauptspeicherauszug sind also die Inhalte aller Hauptspeicherplätze, die zwischen den Plätzen liegen, die durch die Namen der begrenzenden Variablen a_i bzw. b_i spezifiziert sind, eingeschlossen. (Welche Variablen insgesamt davon betroffen werden kann aus den Programmlisten entnommen werden, die während der Übersetzung des Quellenprogrammes gedruckt werden.) Durch die f_i wird das Format angegeben, das für die Ausgabe des Speicherauszuges bestimmend sein soll. Hierzu gibt es die in Abb. 15.9. aufgeführten Möglichkeiten.

f_i	Format
0	Sedezimalziffern
1	LOGICAL*1
2	LOGICAL*4
3	INTEGER*2
4	INTEGER*4
5	REAL*4
6	REAL*8
7	COMPLEX*8
8	COMPLEX*16
9	literalform

Abb. 15.9. Format von Speicherauszügen

Wir sind noch nicht allen Datenformaten begegnet, die in der Abb. 15.9. aufgeführt sind.

7) CALL PDUMP

Die Anweisung CALL PDUMP unterscheidet sich von der Anweisung CALL DUMP nur in der Hinsicht, daß nach dem Speicherauszug die Ausführung des Objektprogrammes mit dem Statement wieder aufgenommen wird, das als nächstes auf die Anweisung CALL PDUMP folgt.
Die Verwendung dieser sowie der zuvor beschriebenen Anweisung CALL DUMP beschleunigt wesentlich die Aufklärung von Programmirrtümern in einem Programm. So befremdend es auch klingen mag: Ein guter Programmierer plant Speicherauszüge bereits beim Schreiben eines Programmes.

Die restlichen Statements, die in diesem Abschnitt besprochen werden, sind Reminiszensen an eine frühere Version der FORTRAN-Sprache, nämlich an die Version II. [1)]

8) CALL EXIT

Dieses Statement besitzt die Form

```
CALL EXIT
```

und ist gleichwertig dem Statement STOP.

9) READ b, liste

Die Anweisung

```
READ b, liste
```

entspricht der Anweisung

```
READ (a, b)liste
```

Für a muß dabei jedoch die Einheitennummer des Systemkartenlesers eingesetzt sein.

10) PUNCH b, liste

Diese Anweisung ist äquivalent der Anweisung

```
WRITE (a, b)liste
```

Für a muß hierbei jedoch die Einheitennummer des Systemkartenstanzers spezifiziert sein.

11) PRINT b, liste

Diese Anweisung ist äquivalent der Anweisung

```
WRITE (a, b)liste
```

Durch a muß hierbei jedoch der Systemdrucker spezifiziert sein.

12) CALL OVERFL(j)

Durch j muß eine Variable des ganzzahligen Typs bezeichnet werden. Dieser wird der Wert 1 zugewiesen, wenn ein Gleitpunktüberlauf eingetreten ist. Die Variable j wird jedoch auf 3 gesetzt, wenn es zu einem Gleitpunktunterlauf gekommen ist, und auf 2, wenn es zu keiner der beiden genannten Bedingungen gekommen ist. Nach Ausführung dieser Anweisung befindet sich der Computer in einem bedingungsfreien Zustand.

---

1) FORTRAN-Kompilierer dieser Version standen bei IBM-Computern der sogenannten 2. Generation zur Verfügung, also beispielsweise für die Typen IBM 1401, IBM 1410, IBM 1620, IBM 7010, IBM 7040, IBM 7044, IBM 7070, IBM 7090 und IBM 7094.

13) CALL DVCHK(j)

Die ganzzahlige Variable j wird auf 1 gesetzt, wenn es zu einem Divisionsfehler (Division durch 0) gekommen ist, anderenfalls wird ihr der Wert 2 zugewiesen. Nach Ausführung dieser Anweisung ist die Fehlerbedingung wieder gelöscht.

14) CALL SLITE(i)

Hierbei ist an Stelle von i entweder 0, 1, 2, 3 oder 4 einzusetzen. Wenn i = 0 ist, so wird die entsprechende Prüfanzeige, d.h. letzten Endes ein interner Programmschalter, auf „ein" gestellt. Ist dagegen i = 0, so werden alle vier Prüfanzeigen gelöscht. Die Stellung einer solchen imaginären Prüfanzeige kann durch das nachfolgend beschriebene Statement abgefragt werden.

15) CALL SLITET(i,j)

Hierbei muß j eine ganzzahlige Variable sein, und i muß gleich 1, 2, 3 oder 4 sein. Die Stellung der Prüfanzeige i wird geprüft; steht sie auf „ein", so wird sie auf „aus" gestellt und außerdem j auf den Wert 1 gesetzt. Im anderen Fall wird j der Wert 2 zugewiesen.

15.7. Eingabe/Ausgabe bei Magnetbändern und Magnetplattenspeichern

Zusätzlich zu den uns bereits bekannten Statements

```
READ
WRITE
END FILE
```

stehen uns in der FORTRAN-Sprache weitere Statements zur Verfügung, die auf Dateien bezogen werden können, die auf Magnetbändern oder Magnetplattenspeichern residieren.

Drei Statements können auf konsekutiv organisierte Dateien angewendet werden. Die erste ist uns bereits bekannt, wir erwähnen sie nur der Vollständigkeit halber noch einmal.

1) END FILE a

Durch a wird eine Einheitennummer spezifiziert; diese entspricht dem a in den dazugehörigen Anweisungen

WRITE(a,b)liste

Bei Magnetbändern wird durch diese Anweisung eine Bandmarke geschrieben, die damit den EOF-Satz [1] anzeigt. Bei Plattenspeichern erfolgt die Aufzeichnung eines EOF-Satzes. Ein nachfolgendes WRITE-Statement mit der gleichen Einheitennummer a definiert damit den Beginn einer neuen Datei, die über die gleiche Einheit ausgegeben wird.

2) REWIND a

Diese Anweisung bewirkt das Zurückspulen des auf der Einheit a eingespannten Magnetbandes. Bei Plattenspeichern werden die Schreib/Lese-Köpfe wieder auf den Anfang der Datei eingestellt, so daß sich nachfolgende READ- bzw. WRITE-Anweisungen auf den ersten Satz der Datei beziehen.

1) EOF - End of File (Dateiende).

3) BACKSPACE a

Durch diese Anweisung werden Dateien auf Magnetbändern um einen Satz zurückgesetzt. Auf ein zurückgespultes Magnetband hat diese Anweisung keinen Einfluß. Bei Plattenspeichern erfolgt ein logisches Zurücksetzen um einen Satz. Bei jeder Einheit hat damit ein anschließendes READ- oder WRITE-Statement den erneuten Zugriff zum zuletzt angesprochenen Satz zur Folge.

Oft werden Magnetbänder bzw. Plattenspeicher als zeitweilige Zwischenspeicher benutzt. Das ist hauptsächlich dann erforderlich, wenn der Hauptspeicherbedarf herabgesetzt werden muß. Die zu diesem Zweck herausgeschriebenen Sätze müssen später in den Hauptspeicher zurückgelesen und im Hauptspeicher wieder so gespeichert werden, wie sie es vor dem Herausschreiben waren. Die Zeit, die man für das Aufbereiten der Sätze benötigt, ist genau so verschwendet wie die Zeit, die für das Entformatisieren nach dem Zurücklesen notwendig ist. Da das Formatisieren und Entformatisieren doch recht erhebliche Zeiten beanspruchen können, gibt es in der FORTRAN-Sprache auch Anweisungen, die das formatfreie Schreiben und Lesen veranlassen. Es handelt sich hierbei um Variationen der bekannten WRITE- bzw. READ-Anweisung. Diese Statements besitzen für diese Fälle den nachstehenden Aufbau:

```
READ  (a)  liste
WRITE (a)  liste
```

Die Daten werden so geschrieben, wie sie im Hauptspeicher verschlüsselt sind, und später direkt wieder in den Hauptspeicher in der gleichen Form eingelesen. Theoretisch können die formatfreien READ- bwz. WRITE-Anweisungen jede EA-Einheit ansprechen, doch ist ihre sinnvolle Anwendung auf Magnetbandeinheiten und auf Einheiten mit direkter Zugriffsmöglichkeit (Plattenspeicher, Trommelspeicher, Magnetstreifenspeicher) begrenzt. Durch a wird bei beiden Einheiten in üblicher Weise die Einheitennummer angegeben. Die Optionen END und ERR können so benutzt werden, wie wir es bei der formatbehafteten EA dargestellt hatten.

Bis hierhin haben wir nur über die Eingabe/Ausgabe von Sätzen gesprochen, zu denen sequentiell zugegriffen wird. Bei Plattenspeichern und bei anderen Speichergeräten mit direkter Zugriffsmöglichkeit ist es jedoch oft wünschenswert, daß man die Sätze direkt ansprechen und verarbeiten kann. Die FORTRAN-Sprache erlaubt auch die direkte Verarbeitung, freilich nur im beschränkten Umfang, d.h. unter ganz bestimmten Voraussetzungen. Jede Datei, deren Sätze direkt angesprochen werden sollen, muß durch eine DEFINE FILE-Vereinbarung gesondert definiert werden. Dieses Statement besitzt folgende allgemeine Form:

DEFINE FILE $a_1(m_1,h_1,f_1,v_1),a_2(m_2,h_2,f_2,v_2)$ ...

Durch jede Spezifikationsgruppe $a_i(m_i, h_i, f_i, v_i)$ wird eine einzige Datei definiert. Das DEFINE FILE-Statement ist ein Spezifikationsstatement und muß im Programm vor allen EA-Anweisungen erscheinen, die die betreffende Datei ansprechen.

Durch die a_i werden Dateinummern vergeben; diese müssen ganze Zahlen ohne Vorzeichen sein. Durch die m_j wird die maximale Anzahl von Sätzen in der betreffenden Datei festgehalten. Die Anzahl der tatsächlich in die Datei aufgenommenen Sätze darf kleiner als m_j sein, jedoch keinesfalls größer. Durch die f_j wird festgelegt, wie die Sätze in die Datei geschrieben werden sollen, d.h. wie sie dort stehen sollen. Drei verschiedene Möglichkeiten existieren hierfür:

a) f_i=E zeigt an, daß die Eingabe/Ausgabe von formatgebundenen Sätzen erfolgt;

b) f_j=U zeigt an, daß alle Sätze in formatfreier Form ausgegeben bzw. eingelesen werden;

c) f_j=L zeigt an, daß bei der Eingabe/Ausgabe sowohl formatgebundene als auch formatfreie Sätze behandelt werden sollen.

Durch die h_j wird die Maximallänge der von den Eingabe/Ausgabe-Anweisungen angesprochenen Sätze angegeben. Ist f_j gleich E oder gleich L, so muß die Länge in Bytes spezifiziert werden, bei f_j gleich U in Worten. Hierbei gilt selbstverständlich die bekannte Relation, daß vier Bytes ein Wort bilden. Die v_j sind nichtindizierte ganzzahlige Variablen, die man als beigeordnete Variablen zu bezeichnen pflegt. Die Sätze in den Dateien werden als numerierte Sätze angesehen, wobei die Nummern von 1 bis einschließlich m_j reichen. Nach jedem READ- bzw WRITE-Statement wird die der betreffenden Datei beigeordnete Variable v_j auf den Wert gesetzt, der der Satznummer des nächstfolgenden Satzes entspricht. Wenn z.B. durch ein READ-Statement der Satz mit der lfd. Nummer 45 gelesen wurde, so erhält die beigeordnete Variable v_j nach Ausführung dieser Anweisung den Wert 46 zugewiesen.

Wir wollen jetzt die Vereinbarung DEFINE FILE an einem Beispiel veranschaulichen:

```
DEFINE FILE  7(256,8,U,KW), 8(66,145,E,LQ)
```

Durch diese Vereinbarung sind zwei Dateien festgelegt. Die Datei 7 besteht aus maximal 256 formatfreien Sätzen von maximal 32 Bytes = 8 Worten Länge. Die beigeordnete Variable KW nimmt die Werte $2,3,\cdots,257$ nach dem Schreiben der Sätze mit den Nummern $1,2,\cdots,256$ an. Die Datei 8 besteht aus 66 Sätzen, wobei jeder nicht länger als 145 Bytes ist; außerdem sind die Sätze an ein bestimmtes Format gebunden. Die Datei 7 könnte z.B. zur vorübergehenden Speicherung der Elemente einer Matrix mit 16 Zeilen und 16 Spalten verwendet werden. In der Datei 8 könnte dagegen eine komplett aufbereitete, aus 66 Zeilen sich zusammensetzende Druckseite untergebracht werden, die später über einen Drucker des Typs IBM 1443 gedruckt werden soll.

Für die Eingabe/Ausgabe von Sätzen stehen die Anweisungen

```
READ  (a'r,b)  liste
WRITE (a'r,b)  liste

READ  (a'r)  liste
WRITE (a'r)  liste
```

zur Verfügung. Mit Ausnahme des hinzugefügten r besitzen sie also den gleichen Aufbau wie die bereits besprochenen READ/WRITE-Anweisungen für Dateien, die den direkten Zugriff zu ihren Sätzen nicht zulassen. Das Anführungszeichen ist stets zusätzlich erforderlich. Durch r wird ein ganzzahliger Ausdruck spezifiziert, der gleich der relativen Satznummer, d.h. der Position des anzusprechenden Satzes innerhalb der Datei ist. Hierfür wird meist die beigeordnete Variable des dazugehörigen Statements DEFINE FILE benutzt, doch kann für r auch jeder beliebige andere Ausdruck eingesetzt werden. Die Anweisung

```
READ (7'KW)  SR
```

bewirkt z.B., daß der 8. Satz aus der Datei 7 formatfrei gelesen wird, falls KW vor Ausführung dieser Anweisung gerade den Wert 8 besitzt; nach Ausführung dieser Anweisung würde KW dann den Wert 9 aufweisen.

Die Anweisung

```
WRITE (8'7*JP-4,1235)  LA,SPD,CDU,CSU,FDP,NPD
```

bewirkt dagegen, daß ein Satz, gebunden an die FORMAT-Vereinbarung 1235, auf den Plattenspeicher geschrieben wird, auf dem die Datei 8 steht. Der Wert des Ausdruckes 7*JP-4 muß hierbei zwischen 1 und 66 liegen, wenn die Datei 8 so definiert worden ist, daß sie aus nicht mehr als 66 Sätzen bestehen darf. Sei, um bei einem Beispiel zu bleiben, JP=3, dann ist 7 JP-4=17. Es wird damit der 17. Satz geschrieben; nach dem Schreiben besitzt dann die beigeordnete Variable LQ den Wert 18. LQ erhält deswegen einen neuen Wert zugewiesen, weil diese Variable in der Vereinbarung DEFINE FILE erwähnt wurde; dabei spielt es überhaupt keine Rolle, ob die einer Datei beigeordnete Variable in einer READ- bzw. WRITE-Anweisung aufgeführt wird oder nicht. Bei Dateien, die durch die Vereinbarung DEFINE FILE dem direkten Zugriff erschlossen werden, kann die Option ERR in READ-Statements so eingesetzt werden, wie wir es für die READ-Anweisung beim sequentiellen Zugriff kennengelernt haben. Die Option END zu verwenden, ist nicht gestattet, da es bei direkten Zugriffen keinen „letzten" Satz einer Datei geben kann.

Schließlich kann noch das Statement

```
FIND (a'r)
```

in einem Programm dazu verwendet werden, um das Suchen zu einem bestimmten Satz einzuleiten. Die Benutzung dieser Anweisung ist nicht erforderlich, doch ein geeigneter Einsatz derselben in einem Programm kann den Zugriff zum nächsten zu lesenden bzw. zu schreibenden Satzbereich wesentlich beschleunigen. Verwendet man eine FIND-Anweisung unmittelbar vor einer READ- bzw. WRITE-Anweisung, so ergibt sich daraus kein Nutzen. Kann jedoch ein FIND-Statement in gebührendem Abstand vor dem zugehörigen READ- bzw. WRITE-Statement codiert werden, so kann der Zugriffsmechanismus der Speicher mit direkter Zugriffsmöglichkeit rechtzeitig auf den angesprochenen Satzbereich eingestellt werden, so daß sich eine geringere Verzögerungszeit ergibt, wenn die Lese- bzw. Schreiboperationen wirklich stattfinden. Die FIND-Anweisung darf für Dateien, die direkte Zugriffe zu ihren Sätzen nicht erlauben, nicht benutzt werden. Beim Mehrprogrammbetrieb (Multiprogramming) braucht dieses Statement die Effizienz nicht unbedingt zu verbessern.

Beim Betriebssystem DOS sind einige Einschränkungen bei der Planung von EA-Operationen zu beachten. Bei starr forlaufend (konsekutiv) organisierten Dateien sind die Satzlängen auf 260 Stellen limitiert. Von mehreren auf einem Magnetband liegenden Datenmengen wird nur die erste als Datei für ein Programm akzeptiert. Datenmengen, die für den direkten Zugriff vorgesehen sind (DEFINE FILE -Statement), müssen zuerst mit Hilfe eines Dienstprogrammes initialisiert werden, bevor sie als Datei in einem vom FORTRAN-Kompilierer generierten Objektprogramm angesprochen werden können. Um dieses Dienstprogramm korrekt einsetzen zu können, ist eine Menge von Steuerstatements in der Jobsteuersprache abzufassen. Meist werden diese wie folgt lauten:

```
// DLBL UOUT,···
// EXTENT ···
// EXEC CLRDSK
// UCL B=(K=Ø,D=Länge),X'ØØ',ØY
// END
```

Der so festgelegte Jobstep [1] muß der Ausführung des Objektprogrammes vorausgehen. Er kann dem Bindeprozeß (Linkage Editor) nachfolgen. Im UCL-Statement darf dem Schlüsselwort UCL jeweils nur ein einziges Leerzeichen vorausgehen bzw. nachfolgen. Im END-Statement darf ebenfalls nur ein einziges Leerzeichen vor dem Schlüsselwort END stehen. Durch das Statement DLBL ist eine Datei zu spezifizieren, die zum Typ SD gehört, d.h. einer starr fortlaufend organisierten Datei, die auf einer SEDZ liegt.

Dateien, die starr fortlaufend organisiert und somit für den sequentiellen Zugriff präpariert sind, können von jedem anderen Programm gelesen werden, wenn sie Standardkennsätze aufweisen. Es spielt dabei überhaupt keine Rolle, ob die Dateien auf Magnetbändern oder auf Plattenspeichern gespeichert sind, und es ist ebenso unwichtig, in welcher Sprache (FORTRAN, PL/1, COBOL, AL, RPG) die Programme geschrieben wurden.

15.8. Die Schleifennotation in Datenlisten von Eingabe/Ausgabe-Anweisungen

In der FORTRAN-Sprache wird uns die Möglichkeit geboten, eine der DO-Anweisung ähnliche Schleifennotation in den Datenlisten von Eingabe/Ausgabe-Anweisungen zu verwenden. Wir wollen diese Möglichkeit an einigen Beispielen demonstrieren.

```
WRITE (6,13)  A,B(1),B(2),B(3),B(4),B(5),B(6)
```

Durch diese Anweisung werden sieben Zahlen herausgeschrieben und zwar so aufbereitet, wie es in der FORMAT-Vereinbarung mit der Nummer 13 festgelegt ist. Eine alternative Methode zur Niederschrift dieser Anweisung ist die folgende:

```
WRITE (6,13)  A,(B(I),I=1,6)
```

Die Größen, die der Indizierung unterliegen, werden hierbei in Klammern eingeschlossen. I stellt die von der DO-Anweisung her bekannte Laufvariable dar, 1 ist ihr Anfangswert und 6 ihr Endwert.

Nach diesem Beispiel wollen wir einige andere Beispiele verfolgen, bei denen wir die beiden Schreibweisen einander gegenüberstellen (siehe Abb. 15.10.).

```
1. a) WRITE (6,13)  A,B(1),B(2),B(3),B(4),D(1),D(2),D(3),D(4)
   b) WRITE (6,13)  A,(B(I),I=1,4),(D(I),I=1,4)

2. a) WRITE (6,13)  A,B(1),D(1),B(2),D(2),B(3),D(3),B(4),D(4)
   b) WRITE (6,13)  A,(B(I),D(I),I=1,4)
```

Abb. 15.10. Beispiele für die Schleifennotation in Datenlisten

Bei mehr als einem Index ist die Situation ähnlich, es braucht nur ein weiteres Klammernpaar benutzt zu werden. Wenn wir annehmen, daß KO bzw. K im Programm definierte ganzzahlige Variablen sind, dann können wir auch die Beispiele leicht verfolgen, die in Abb. 15.11. aufgeführt sind.

1) Unter Länge ist die Satzlänge zu verstehen.

```
1. a) WRITE (6,13)  B(KO),KO,A,E(1,1),E(1,2),E(1,3),···,E(1,KO)
   b) WRITE (6,13)  B(KO),KO,A,(E(1,J),J=1,KO)

2. a)   WRITE (6,14)  E(1,1),E(2,1),E(3,1),E(4,1),E(5,1),E(6,1),U(1),
       1              E(1,2),E(2,2),E(3,2),E(4,2),E(5,2),E(6,2),U(2),
       2              ···
       ⋮              ⋮
       K              E(1,K),E(2,K),E(3,K),E(4,K),E(5,K),E(6,K),U(K)
   b)   WRITE (6,14)  ((E(I,J),I=1,6),U(J),J=1,K)

3. a) READ (5,15)  H(1,1),H(2,1),H(3,1),H(1,2),H(2,2),H(3,2)
   b) READ (5,15)  ((H(I,J),I=1,3),J=1,2)
```

Abb. 15.11. Weitere Beispiele für die Schleifennotation in Datenlisten

Wenn im Beispiel 3. der Bereich H durch

```
DIMENSION  H(3,2)
```

vereinbart wurde, so können wir uns in der READ-Anweisung auch auf den gesamten Bereich dadurch beziehen, daß wir einfach nur

```
READ (5,15)  H
```

schreiben. Diese Form kann aber grundsätzlich nur dann verwendet werden, wenn in einer Eingabe/Ausgabe-Anweisung ein gesamter Bereich angesprochen werden soll.

Auch bei der Schleifennotation in den Datenlisten von Eingabe/Ausgabe-Anweisungen braucht der Erhöhungswert (das Inkrement) der Laufvariablen nicht auf 1 beschränkt zu werden.

Betrachten wir deshalb auch für diesen Fall ein Beispiel: Die Schreibweise

```
READ (5,16)  A(1),A(4),A(7),A(1Ø),A(13),A(16)
```

kann durch die Schreibweise

```
READ (5,16)  (A(K),K=1,16,3)
```

ersetzt werden

Einer der Hauptvorteile der Schleifennotation in Datenlisten ergibt sich daraus, daß der Programmierer in die Lage versetzt wird, eine variable Anzahl von zu lesenden bzw. zu schreibenden Daten anzugeben, da die in der Schleifennotation auftretenden Größen auch Variable sein können. Er braucht sich also nicht immer auf den ganzen Bereich zu beziehen. Der zweite Vorteil liegt auf der Hand: Die Schreibweise ist knapper und zugleich auch übersichtlicher.

Die Schleifennotation in den Datenlisten bei den Engabe/Ausgabe-Anweisungen wird von den Kompilierern genau so behandelt wie die DO-Anweisungen selbst. Es gelten also auch für sie die Regeln, die wir bei der Besprechung der DO-Anweisungen kennengelernt haben.

15.9. Namenslisten

Die Statements, die wir in diesem Abschnitt behandeln wollen, gelten nur für die volle FORTRAN-Sprache.

Mitunter ist es äußerst sinnvoll, wenn man Daten einlesen kann, die auf einer Karte (bzw. auf Karten) in der folgenden Form abgelocht sind:

```
KL=4, AWR=3.45E-Ø3, WR=5.6,1.4,1.3,2.4,3.4,6.7
```

Hierbei wird durch KL eine ganzzahlige Variable, durch AWR eine reelle Variable und durch WR ein Bereich mit zwei Dimensionen bezeichnet (WR muß dazu im Programm in einem passenden DIMENSION-Statement erscheinen). Um die Eingabe/Ausgabe von Daten in dieser Form zu ermöglichen, müssen im Programm die nachfolgenden Statements erscheinen:

```
NAMELIST    /x/a1,a2,···,ak/y/b1,b2,···,bn/z/···
READ (a,x)   bzw.   READ (a,y)   bzw.   READ (a,z)
WRITE (a,x)  bzw.   WRITE (a,y)  bzw.   WRITE (a,z)
```

Wie üblich stellt a hierbei die Nummer der betreffenden EA-Einheit dar, während x, y, z usw. sogenannte NAMELIST-Namen darstellen. Die a_i bzw. die b_i sind die Namen von Variablen bzw. Bereichen, die zu der speziellen Namensliste gehören. NAMELIST-Namen müssen genau so gebildet werden wie alle anderen Namen eines FORTRAN-Programmes. Sie unterliegen außerdem noch folgenden Bedingungen:

1) Der Name kann nur in einem einzigen und in keinem weiteren NAMELIST-Statement erscheinen.

2) Der Name kann in einem Programm sonst nur noch in EA-Anweisungen auftreten.

In der Vereinbarung NAMELIST muß ein NAMELIST-Name in Schrägstriche eingeschlossen werden. Die Namen der zu einer speziellen Namensliste gehörenden Variablen bzw. Bereiche müssen hinterher geschrieben werden; die Reihenfolge ist dabei gleich, Hauptsache, sie sind durch Kommas voneinander getrennt. Es dürfen soviele verschiedene Namenslisten verwendet werden, wie es nur immer für wünschenswert erachtet wird, doch kein NAMELIST-Name darf mehr als eine Namensliste bezeichnen. Die Namen von Variablen bzw. Bereichen dürfen dagegen in mehr als einer Namensliste erscheinen.

Bei den READ- bzw. WRITE-Anweisungen wird keine Datenliste niedergeschrieben; auch wird keine Bezugnahme auf eine FORMAT-Vereinbarung angegeben. Stattdessen wird nur der NAMELIST-Name codiert. Für die Eingabe gelten folgenden Regeln:

a) Die Daten müssen auf dem Eingabemedium in der folgenden Form erscheinen:

   variablenname=konstante [1)]

   Der Variablenname darf indiziert sein.

b) Die zu Bereichen gehörenden Daten müssen in der folgenden Form auf dem Eingabemedium erscheinen:

   bereichsname=konstante$_1$,k*konstante$_2$,··· [1)]

   Hierbei bedeutet k eine ganze Zahl ohne Vorzeichen; durch ihre Angabe ist festgelegt, daß die nachfolgende Konstante k-mal zu wiederholen ist, d.h. k aufeinander-

1) Die Formen werden im weiteren Text dieses Abschnittes als „Zuweisungen" bezeichnet.

folgenden Bereichselementen als Wert zugewiesen werden soll. Es müssen genau so viele Konstanten spezifiziert werden, wie der Bereich Elemente ausweist.

c) Bei allen Eingabesätzen wird die erste Spalte ignoriert.

d) Die Spalte 2 des ersten Eingabesatzes muß das Zeichen & enthalten. Der NAMELIST-Name muß ab Spalte 3 erscheinen. Weitere Informationen darf dieser erste Satz nicht enthalten.

e) In den nachfolgenden Sätzen dürfen beliebig vielen Variablen bzw. Bereichen Werte zugewiesen werden. Es dürfen jedoch nur die Variablen aufgeführt werden, deren Namen in die entsprechende Namensliste aufgenommen sind (siehe NAMELIST-Vereinbarung). Jeder Zuweisung muß unbedingt ein Komma folgen. In jedem Satz muß mindestens eine Zuweisung erfolgen. Leerzeichen dürfen zur Verbesserung der Lesbarkeit zusätzlich in die Sätze eingefügt werden, doch dürfen sie keinesfalls in Namen oder Konstanten eingebettet werden.

f) Nach der letzten Zuweisung im letzten Datensatz ist kein Komma mehr erforderlich. Ein evtl. doch an dieser Stelle erscheinendes Komma wird jedoch nicht als Fehler angesehen.

g) Nach dem letzten Datensatz muß ein weiterer Eingabesatz folgen, der auf den Spalten 1 bis 5 die folgenden Informationen enthält:

```
b&END
```

(b bedeutet hier das Leerzeichen.)

h) Innerhalb der Datensätze dürfen den Variablen Werte in einer beliebigen Reihenfolge zugewiesen werden.

Bei der Ausgabe wird das gleiche Grundformat verwendet wie bei der Eingabe. Somit können die ausgegebenen Sätze später wieder als Eingabesätze dienen. Es werden jedoch Zuweisungen für alle in der betreffenden Namensliste aufgeführten Variablen bzw. Bereiche ausgegeben. Eine typische Ausgabe sieht wie folgt aus: [1)]

```
b&XYZ
bA=4.567,4.589,3.468,1.423,B=567.13,I=768
b&END
```

Die Ausgabe dieser drei Sätze könnte z.B. von folgender Statementfolge her stammen:

```
DIMENSION  A(4)
NAMELIST   /XYZ/A,B,I
WRITE (6,XYZ)
```

Bei der Ausgabe wird eine Formatangabe des G-Typs verwendet. Für diese sind soviele Ziffernstellen vorgesehen, daß mit Sicherheit keine gültigen Ziffern verloren gehen können.

Man sollte immer auf den Unterschied zwischen den beiden Anweisungen

```
WRITE (a,x)          und
WRITE (a,b)
```

achten. Im letzteren Fall ist auf die Angabe einer Datenliste verzichtet worden. Deshalb werden bei ihr auch keine Werte von Variablen ausgegeben. Die einzige Ausgabe, die durch sie hervorgerufen wird, ist entweder ein Leersatz oder ein Satz, der nur aus

1) b bedeutet hier das Leerzeichen.

Hollerithinformationen besteht. Welche der beiden Möglichkeiten eintritt, hängt ausschließlich von der zugehörigen FORMAT-Vereinbarung b ab.

Zum Schluß dieses Abschnittes wollen wir darauf hinweisen, daß das Statement BACKSPACE nicht auf Dateien bezogen werden darf, für die die EA über Namenslisten vorgesehen ist.

Aufgabengruppe 41

1) Es sei

$$a_n = \frac{(n+1)^n}{n^{2n}} \qquad \text{für } n = 1, 2, \cdots$$

Es sei weiterhin

$$s_n = a_1 + a_2 + \cdots + a_n$$

Schreibe ein Programm, das eine Tabelle mit den Werten von n, a_n und s_n für $n = 1, 2, \cdots$ ausdruckt. Die Ausgabe soll beendet werden, wenn $a_n < 10^{-7}$ ist.

2) Aufgabe 1) ist wie folgt abzuändern: Auf die linke Hälfte einer Zeile soll das Wertetripel n, a_n, s_n für ungerade n, auf die rechte Hälfte derselben Zeile für gerade n gedruckt werden. Auf diese Weise kann die Zahl der Ausgabezeilen auf die Hälfte reduziert werden.

3) Schreibe ein Programm zur Berechnung der Summe von

$$1 - \frac{1}{2} - \frac{1}{3} + \frac{1}{4} + \frac{1}{5} - \frac{1}{6} - \frac{1}{7} ++ -- \cdots$$

4) Es soll die Reihe

$$\frac{\sin x}{1} + \frac{\sin(3x)}{\sqrt{3}} + \frac{\sin(5x)}{\sqrt{5}} + \cdots$$

für $x = 1^o, 2^o, \cdots, 180^o$ berechnet werden. Zur Aufgabe sollen gelangen:

a) Der Wert von x, ausgedrückt in Graden,

b) Die Anzahl der Glieder, die zur Berechnung herangezogen werden müssen, damit die Summe der Glieder bis auf 7 Dezimalstellen genau ist,

c) Die Summe der Glieder.

5) Was geschieht, wenn wir die gleichen Berechnungen wie bei der Aufgabe 4) mit der Reihe

$$\frac{\sin x}{1} + \frac{\sin (3x)}{3} + \frac{\sin (5x)}{5} + \cdots$$

ablaufen lassen?

6) Programmiere die Aufgabe 5) der Aufgabengruppe 32 neu; verwende dabei in der Datenliste die Bereichsschreibweise!

7) Programmiere die Aufgabe 5) der Aufgabengruppe 32 neu; verwende dabei für die Elemente der Datenliste die Indexschreibweise!

8) Programmiere unter Verwendung der Rechensprunganweisung die Aufgabe 4) der Aufgabengruppe 31 neu!

9) Schreibe ein Programm zur Berechnung der Werte von g(x) im Wertebereich von x = 0 bis x = 0.99 bei einer Schrittweite von $\Delta x = 0,01$!

Es ist

$$g(x) = \int_0^x \frac{1}{\pi} \ln\left(\frac{1-t}{1+t}\right) \frac{dt}{t}$$

$$= \frac{2}{\pi} \sum_{k=0}^{\infty} \frac{x^{2k-1}}{(2k-1)^2}$$

Nach jedem Rechenschritt sind jeweils die Werte von x in der Form 0.xx und von g(x) in der Form x.xxxxxx auszugeben.

10) Führe die gleichen Berechnungen und Ausgaben wie bei Aufgabe 9) durch und vergleiche die Resultate mit den Ergebnissen von Aufgabe 9)!

Es ist

$$h(x) = \frac{\pi}{4} - \frac{2}{\pi} \sum_{k=1}^{\infty} \frac{1-(2k-1)\ln\left(\frac{1-x}{1+x}\right)}{(2k-1)^2} \left(\frac{1-x}{1+x}\right)^{2k-1}$$

11) Unter Benutzung der Ergebnisse der Aufgaben 9) und 10) soll eine Tabelle gedruckt werden, die die x-Werte und die Anzahl der Glieder enthält, die zur Bestimmung der Funktionswerte erforderlich waren.

12) Schreibe ein Programm zur Berechnung von

$$s = \sum_{j=1}^{m} \left[\sum_{i=1}^{j} (3j + ij)^{i+1} \right]^2$$

für $m = 1, 2, \cdots, 20$. Ausgegeben werden sollen die Werte von m und die dazugehörigen Werte von s. Für m und auch für s sollen dabei F-Formatangaben verwendet werden. Überschreiten die Werte von s jedoch den Betrag von 10^{12}, so sollen die s-Werte unter Verwendung der E-Formatangabe ausgegeben werden.

13) Programmiere Aufgabe 12) so um, daß keine DO-Anweisung mehr auftritt!

16. Fortgeschrittene FORTRAN-Sprachelemente

In die beiden Betriebssysteme DOS und OS sind Kompilierer für die volle FORTRAN-Sprache integriert, so daß wir bei der Besprechung eigentlich keine Rücksicht auf die einzelnen Sprachebenen zu nehmen brauchen. Wenn wir es dennoch tun, so geschieht das unter dem Aspekt der Kompatibilität, unter dem Aspekt des Programmaustausches mit anderen Rechenzentren. Und möglicherweise sind dort Computer installiert, für die nur Kompilierer für BASIC FORTRAN zur Verfügung stehen.

16.1. Komplexe Variablen

Dieser Abschnitt gilt nicht für die BASIC FORTRAN - Sprache.

In der FORTRAN-Sprache wird eine komplexe Zahl durch ein wohlgeordnetes Paar von reellen Zahlen dargestellt. Komplexe Konstanten werden in der Form

(x_r, x_i)

geschrieben; hierbei sind x_r und x_i reelle Konstanten. Der Realteil der komplexen Zahl wird durch x_r, der Imaginärteil durch x_i dargestellt. Es können hierbei beliebige gültige reelle Konstanten niedergeschrieben werden, eine Bedingung muß allerdings erfüllt sein: Die beiden Konstanten müssen die gleiche Länge aufweisen. Wird bei der Niederschrift der beiden Konstanten das Exponentenfeld durch ein D eingeleitet oder weisen beide Konstanten mehr als sieben gültige Ziffern auf, so werden sowohl der Realteil als auch der Imaginärteil als erhöhtgenaue (doppelt genaue) Konstanten gespeichert (sie belegen also ein Doppelwort, d.h. 8 Bytes). Eine solche komplexe Konstante nimmt also insgesamt 16 Bytes ein.

Komplexe Variablen können mittels expliziter Typvereinbarungen festgelegt werden. Diese treten in der Form

```
COMPLEX*8  v1,v2,...          oder in der Form
COMPLEX*16 v1,v2,...
```

auf. Verzichten wir in einer Typvereinbarung auf die Längenangaben *8 bzw. *16, steht also COMPLEX allein da, so ist eine solche Schreibweise der Codierung

```
COMPLEX*8
```

äquivalent. Zur Definition komplexer Variablen darf selbstverständlich auch die IMPLICIT-Vereinbarung eingesetzt werden, die dann, um bei einem Beispiel zu bleiben, in der Form

```
IMPLICIT  COMPLEX*16(A), COMPLEX(B-C)
```

geschrieben wird. Wenn wir dieser Vereinbarung noch die Vereinbarung

```
COMPLEX*8  BK35Ø, IFA*16, MZ(5,6), A
```

folgen lassen, so können wir nachstehendes Resümee ziehen: Alle Variablen, deren Namen mit A beginnt, werden als komplexe Variablen mit der Länge 16 angesehen, während diejenigen, deren Namen mit B oder C anfangen, als komplexe Variablen mit der Länge 8 betrachtet werden. Die Variablen BK35Ø, IFA und der aus 30 Elementen bestehende Bereich MZ gehören ebenfalls zum komplexen Typ; ihnen ist z.T. die Länge 8 (BK35Ø und MZ), z.T. auch die Länge 16 (IFA) zugeordnet. Der Variablen A ist schließlich durch die explizite Typvereinbarung ebenfalls eine Länge von 8 zugeordnet, die damit die implizite Festlegung der Länge 16 außer Kraft setzt. Es sollte der Klarheit

wegen noch einmal ausdrücklich betont werden, daß die Variable A nichts mit der Menge von Variablennamen zu tun hat, die mit dem Buchstaben A anfangen. Da sich grundsätzlich eine explizite Typfestlegung über die implizite hinwegsetzt, besitzt die Variable A eben nur eine Länge von acht Stellen, alle anderen Variablen dagegen, deren Namen mit A beginnt, nach wie vor eine Länge von 16 Bytes.

Bei der Bildung arithmetischer Ausdrücke dürfen komplexe Konstanten und Variablen zusammen mit reellen bzw. ganzzahligen Größen verwendet werden. Aus der Berechnung eines solchen Ausdruckes ergibt sich jedoch grundsätzlich ein komplexes Ergebnis. Das Resultat besitzt überdies eine Länge von 16, wenn nur irgendeine der im Ausdruck stehenden Größen diese Länge aufweist oder wenn in den Ausdruck eine reelle Größe der Länge 8 eingeht. In allen anderen Fällen wird ein komplexer Wert der Länge 8 gebildet. Betrachten wir dazu einige Beispiele. In einem Programm sind die beiden folgenden Typfestlegungen vorgenommen worden:

```
REAL*8    AN,NE*4
COMPLEX*8   MA,RI*16
```

Die Verwendung dieser Variablen wird in Abb. 16.1. an einigen Beispielausdrücken demonstriert; die Abb. 16.1. hält zusätzlich die Länge des Resultates fest.

Beispiel	Ausdruck	Ergebnis		
		Typ		Länge
1	AN + MA	komplex	erhöht genau	16
2	NE - 2*MA	komplex	einfach genau	8
3	NE + (3.4D2,-2.4D-2)	komplex	erhöht genau	16
4	RI / MA	komplex	erhöht genau	16
5	NE - RI**2	komplex	erhöht genau	16

Abb. 16.1. Resultate der Berechnung von arithmetischen Ausdrücken

Die Ergibtanweisung

```
MA = AN
```

bewirkt, daß der der reellen Variablen AN zugewiesene Wert in eine komplexe Zahl einfacher Genauigkeit umgewandelt wird, deren Imaginärteil gleich Null ist; diese Zahl wird dann der Variablen MA als Wert zugewiesen.

Bei der Verwendung komplexer Größen sind gewisse Einschränkungen zu berücksichtigen. Potenzen, deren Basis eine komplexe Größe ist, können dann und nur dann gebildet werden, wenn der Exponent eine ganzzahlige Größe ist. Reelle oder gar komplexe Exponenten sind nicht erlaubt. Somit ist die Niederschrift von MA**J gestattet, die von MA**AN, AN**MA und K**RI dagegen nicht. Zu den soeben vorgestellten Ausdrücken ist noch zu bemerken, daß wir J und K implizit als Variablen des ganzzahligen Typs angesehen haben. Wir sollten noch erwähnen, daß Potenzen mit komplexen Exponenten auch dann nicht gebildet werden dürfen, wenn die Basis nicht komplex ist. Der Ausdruck K**RI ist hierfür ein typisches Beispiel.

In IF-Anweisungen dürfen komplexe Ausdrücke nicht auftreten.

Einige der im Anhang F aufgeführten Unterprogramme sind speziell für die Verwendung komplexer Größen geschaffen worden.

Bei EA-Operationen wird eine komplexe Variable so behandelt, als ob zwei aufeinanderfolgende reelle Variablen in der Datenliste vorhanden wären. Somit bewirkt die Statementfolge

```
      COMPLEX C
      WRITE (6,6ØØ1)  C
6ØØ1  FORMAT (2E16.8)
```

daß zunächst der Realteil und Imaginärteil von C, beide im Format E16.8, ausgegeben werden. Der Grund dafür, daß in der vollen Sprache innerhalb von FORMAT-Vereinbarungen ein zweites ineinandergeschachteltes Klammernpaar erlaubt ist, ist wohl in erster Linie darin zu suchen, daß damit die Möglichkeit geboten wird, Gruppen von Basisgruppen zu bilden. Basisgruppen benötigt man aber, um die beiden Werte komplexer Variablen spezifizieren zu können. Verfolgen wir deshalb, um uns zu einem Verständnis dieser Angelegenheit durchringen zu können, die nachstehende Statementfolge:

```
      WRITE (7,124)   J, (CPL(I),I=1,6),KR(J),KR(J+1),
     1                   (CQL(I),I=1,6),MR(J),MR(J+1)
  124 FORMAT (I7/(6(F7.2,'+',F7.2,'I'),2I8))
```

Durch die WRITE-Anweisung werden in Verbindung mit der zugehörigen FORMAT-Vereinbarung drei Zeilen gedruckt: Auf der ersten Zeile erscheint der Wert von J, die zweite Zeile enthält sechs komplexe Zahlen, denen sich zwei ganze Zahlen anschließen, und die dritte Zeile weist den gleichen Aufbau wie die zweite Zeile auf.

16.2. Logische Daten und Statements

Die in diesem Abschnitt zu besprechenden Sprachelemente besitzen nur für die volle FORTRAN-Sprache Gültigkeit.

In der FORTRAN-Sprache versteht man unter logischen Größen solche, die nur zwei verschiedene Werte annehmen können, entweder „wahr" oder „falsch". Die beiden möglichen logischen Konstanten müssen wie folgt codiert werden:

```
. TRUE.
. FALSE.
```

Die beiden Begrenzungspunkte sind notwendiger Bestandteil der Konstanten.

Die logischen Variablen müssen entweder implizit durch die IMPLICIT-Vereinbarung oder explizit durch die LOGICAL-Vereinbarung (eine Typvereinbarung) definiert werden. Betrachten wir dazu zwei Beispiele:

```
IMPLICIT  LOGICAL(L), LOGICAL*1(M)
LOGICAL*4  DAG, RAR*1
```

Logischen Variablen werden entweder 4 Bytes Speicherplatz oder nur 1 Byte zugeordnet; keine andere Längenangabe ist möglich. Die obigen Statements legen deshalb fest, daß z.B. DAG, LOK, LIEBE und LUST logische Variablen der Länge 4 sind, RAR, MEER und MOLLI logische Variablen der Länge 1.

Sechs Vergleichsoperatoren können verwendet werden (Abb. 16.2.).

Die Begrenzungspunkte sind notwendiger Bestandteil der Operatoren. Diese Operatoren können dazu verwendet werden, um arithmetische Bedingungen mit reellen oder ganzzahligen Größen zu untersuchen, das Ergebnis einer solchen Bedingung ist entweder „wahr" oder „falsch". Komplexe Größen dürfen in derartige Vergleichsbedingungen nicht eingehen.

Operator	Bedeutung
. GT.	> (größer als)
. GE.	≧ (größer als oder gleich)
. LT.	< (kleiner als)
. LE.	≦ (kleiner als oder gleich)
. EQ.	= (gleich)
. NE.	≠ (nicht gleich)

Abb. 16.2. Vergleichsoperatoren

Wenn A=7.Ø und B=9.5 ist, so ergibt sich bei Auswertung des Vergleichsausdruckes

```
A .LT. B
```

der Wert .TRUE. . Ist jedoch A=9.5 und B=3.1, so ergibt sich als Wert des Ausdruckes der Wert .FALSE. . Durch die Ergibtanweisung

```
LWX = A .LT. B
```

wird der logischen Variablen LWX der Wert .TRUE. oder der Wert .FALSE. zugewiesen, je nachdem, ob der rechts vom Ergibtzeichen aufgeführte Ausdruck zur Aussage „wahr" oder „falsch" geführt hat. Die sechs Vergleichsoperatoren (siehe Abb. 16.2.) können nicht auf logische oder komplexe Größen angewendet werden. In Verbindung mit ganzzahligen oder reellen Größen gebraucht, führen sie zu einem logischen Wert.

In der FORTRAN-Sprache sind außerdem drei logische Operatoren definiert (siehe Abb. 16.3.).

Operator	Bedeutung
.NOT.	¬ (nicht)
.AND.	∧ (und)
.OR.	∨ (inklusives oder)

Abb. 16.3. Logische Operatoren

Für die logische Beziehung „exklusives oder" ist kein Operator festgelegt worden. Logische Operatoren können nur im Zusammenhang mit logischen Variablen, logischen Konstanten und logischen Ausdrücken gebraucht werden. Betrachten wir einmal den folgenden logischen Ausdruck

```
LWX .AND. .NOT. L
```

Dieser führt auf den Wert .TRUE., wenn LWX den Wert .TRUE. und L den Wert .FALSE. besitzt. Nur dann dürfen zwei logische Operatoren hintereinander geschrieben werden, wenn der zweite Operator .NOT. ist (wie im obigen Beispiel). Da

```
A .LT. B
```

einen logischen Ausdruck darstellt, kann dieser ebenfalls mit einem logischen Operator verknüpft werden. Somit ist z.B.

```
LWX .OR. (A .LT. B)
```

ein gültiger logischer Ausdruck.

Eine Ergibtanweisung in der Form

a=b

wird als logische Ergibtanweisung bezeichnet, wenn a eine logische Variable und b einen gültigen logischen Ausdruck darstellt.

Verzichtet man auf die Verwendung von Klammern, um die Reihenfolge der Operationen innerhalb eines logischen Ausdruckes festzulegen, so werden die einzelnen Operationen in einer wohldefinierten Reihenfolge ausgeführt, d.h. aber nichts anderes, als daß alle Operatoren einer bestimmten Rangordnung unterliegen. Es gilt die in der Abb. 16.4. aufgeführte Rangordnung.

Rangstufe		Operator
(höchste)	1	Berechnung von Funktionswerten
	2	**
	3	* und /
	4	+ und -
	5	.LT. , .LE. , .GT. , .GE. , .EQ. , .NE.
	6	.NOT.
	7	.AND.
(niedrigste)	8	.OR.

Abb. 16.4. Rangordnung der Operatoren

In eine arithmetische IF-Anweisung dürfen keine logischen Ausdrücke aufgenommen werden. Es ist jedoch eine logische IF-Anweisung definiert, die wie folgt aufgebaut sein muß:

IF (b) s

Hierbei bedeutet b irgendeinen logischen Ausdruck und s ist irgendein Statement mit Ausnahme einer Vereinbarung oder einer DO-Anweisung oder eines anderen logischen IF-Statements. Ergibt sich bei der Auswertung des Ausdruckes b, daß dieser gleich .TRUE. ist, so wird die Anweisung s ausgeführt, anderenfalls wird sie übergangen.

Zum Beispiel sind die beiden Anweisungen

```
IF  (LWX .AND. .NOT. L)  J=Ø    bzw.
IF  (LRU)  GO TO 1786
```

gültige logische IF-Anweisungen, wenn alle auftretenden Variablen als logische Variablen erklärt sind.

Es sollte noch vermerkt werden, daß eine logische IF-Anweisung durchaus als letzte Anweisung der Schleifenspanne einer DO-Anweisung erscheinen darf; für arithmetische IF-Anweisungen ist das bekanntlich nicht erlaubt.

Für die Werte von logischen Variablen, auf die in EA-Anweisungen bezug genommen wird, muß eine spezielle Formatangabe in der zugehörigen FORMAT-Vereinbarung stehen. Die für logische Variablen gültige Formatangabe lautet:

Lw

Hierbei bedeutet w wie üblich die Feldlänge. Bei der Eingabe wird das betreffende Feld von links nach rechts auf die Buchstaben T bzw. F durchsucht. Das erste T bzw. das erste F, das auf diese Weise gefunden wird, veranlaßt die Zuweisung des Wertes .TRUE.

bzw. des Wertes .FALSE. zu der entsprechenden Variablen in der Datenliste. Wird weder ein T noch ein F entdeckt, so wird der Variablen der Wert .FALSE. zugewiesen. Bei der Ausgabe wird in das Feld rechtsbündig ein T oder ein F hineingestellt; der Rest des Feldes wird mit Leerzeichen aufgefüllt. Betrachten wir hierzu die Statementfolge von Abb. 16.5.

```
      LWR = .TRUE.
      LN = .FALSE.
      WRITE (7,7Ø7)  LWR,LN,LWR
7Ø7   FORMAT (3L4)
```

Abb. 16.5. Beispiel für eine Ausgabeanweisung mit logischen Variablen

Bei der Statementfolge von Abb. 16.5. kommt es zu der folgenden Ausgabe:

```
bbbTbbbFbbbT
```

Hierbei bedeutet b das Leerzeichen.

Aufgabengruppe 42

1) Schreibe ein Programm, das Karten einliest, auf denen die Werte der Koeffizienten der quadratischen Gleichung $ax^2 + bx + c = Ø$ als komplexe Zahlen im Format E13.6 abgelocht sind. Die Lösungen sind als komplexe Zahlen im gleichen Format auszugeben.

2) Schreibe ein Programm, das n-te Einheitswurzeln berechnet und anschließend ausdruckt. Die n-te Einheitswurzel ist durch

$$\cos t + i \cdot \sin t$$

gegeben, wobei $t=2\pi k/n$ und $k=0,1,2,\cdots,n-1$ ist. Der Wert von n soll eingegeben werden können.

3) Zur Eingabe sollen komplexe Zahlen im Format E18.8 für a, b, c, d, e und f gelangen. Für jede aus diesen sechs Zahlen bestehende Menge werden drei achtzigspaltige Lochkarten vorgesehen. Die Spalten 1 bis 4 dieser Karten enthalten eine Ordnungsnummer, damit die betreffende Zahlenmenge eindeutig identifiziert werden kann. Auf den Spalten 5 bis 8 sind Einsen abgelocht, wenn die Lochkarte die Real- und Imaginärteile von a und b enthält, Zweien bei c und d sowie Dreien bei e und f. Nach dem Einlesen sind die Eingabewerte auszudrucken. Anschließend ist

$$\frac{g(x)}{h(x)} = \frac{ax^3 + bx^2 + cx + d}{cx + f}$$

zu berechnen. Die sich für g(x) und h(x) ergebenden Werte sind zu drucken, desgleichen auch der Quotient und der Rest.

4) Es ist eine Lochkarte einzulesen, die die Werte von n und r enthält. Danach ist eine Tabelle zu drucken, die die Werte von x und die Werte der Potenz $(1+ir)^x$ enthält. Die unabhängige Variable soll dabei den Wertebereich von $1,2,\cdots,n$ durchlaufen. Der Wert von r ist reell und liegt im Format F8.6 vor; n ist eine natürliche Zahl kleiner als 100 (Format I2).

5) Schreibe ein Programm, das Lochkarten einliest, die auf den Spalten 1 bis 6 und 7 bis 12 zwei aus je sechs Buchstaben bestehende Wörter enthalten. In beiden Wörtern sollen keine Buchstaben mehr als ein Mal auftreten dürfen. Über die Konsole

sind zunächst die beiden eingelesenen Wörter auszugeben, danach die Anzahl der Buchstaben, die sowohl im ersten als auch im zweiten Wort vorhanden sind. Die gewünschte Ausgabe sei an einem Beispiel demonstriert:

Bei Eingabe von PINSEL und PUHLER soll die Zeile

```
PINSEL---PUHLER---3
```

ausgegeben werden.

6) Erweitere das Programm von Aufgabe 5) so, daß nach der Eingabe sofort beide Wörter auf sich wiederholende Buchstaben durchsucht werden. Werden solche entdeckt, so ist eine Hinweismeldung auszugeben.

7) Es soll eine Tabelle über den Wahrheitsgehalt D für jede mögliche Kombination von Werten der Variablen A, B und C gedruckt werden. Schreibe ein Programm, das die Tabelle für

```
D = A .OR. .NOT. B .AND. C
```

aufstellt und ausgibt. Die Tabelle soll in der folgenden Form erscheinen:

```
A  B  C     D
=============

T  T  T     T
T  T  F     T
.
.
.
F  F  T     T
F  F  F     F
```

Bei drei Variablen erhalten wir eine aus $2^3=8$ Zeilen bestehende Ausgabeliste.

8) Löse das gleiche Problem wie bei Aufgabe 7) für die folgenden logischen Ausdrücke!

a) `E = A .OR. (B .AND. (C .OR. .NOT. D))`

b) `D = A .OR. (B .AND. .NOT. C)`

c) `D = A .AND. .NOT. B .OR. (C .AND. B)`

d) `C = (A .OR. B) .AND. .NOT. B .OR. A .AND. B`

9) Jede Dezimalziffer kann im BCD-Code[1]) als Kombination von Bits mit den Wertigkeiten 8, 4, 2 und 1 dargestellt werden. Den zur Aufnahme dieser Bits nötigen Bitstellen sollen die Bereichselemente A(1), A(2), A(3), und A(4) zugeordnet werden. A(i) soll auf T gesetzt werden, wenn die BCD-Darstellung der Ziffer auf der Bitstelle mit der Wertigkeit 2^{i-1} eine 1 erfordert, anderenfalls auf F. Unter diesen Voraussetzungen können wir jede Dezimalziffer durch den Wahrheitsgehalt der Elemente des Bereiches A verschlüsseln. Für die Ziffer 9 ergibt sich somit z. B.:

`A(1) = T;  A(2) = F;  A(3) = F;  A(4) = T`

Schreibe ein Programm, das eine Dezimalziffer im Format I1 einliest und das die zugehörigen logischen Werte für die Bereichselemente von A ausgibt.

1) BCD — Binary Coded Dezimal (binär verschlüsselte Dezimalziffer)

10) Ändere das Programm von Aufgabe 9) so ab, daß der Bereich A durch ein 5. Element A(5) ergänzt wird. Dieses Element ist auf T zu setzen, wenn eine gerade Anzahl der übrigen Bereichselemente auf T gesetzt wurde, anderenfalls auf F. Die Gesamtzahl der auf den logischen Wert T gesetzten Bereichselemente soll also stets ungerade sein.

11) Schreibe ein Programm, das Karten einliest, die auf den ersten vier Spalten (Format 4L1) die logischen Werte für die vier Bereichselemente A(1), A(2), A(3) und A(4) enthält. Ausgegeben werden sollen neben den Eingabewerten die logischen Werte von B(1), B(2), B(3), B(4) und von C. Die Werte von B ergeben sich durch die Addition von 1 auf die Werte von A, d.h. A und B sollen als Dualzahlen mit 4 Bitstellen aufgefaßt werden und den logischen Werten T bzw. F sollen die Dualzahlen 1 bzw. 0 entsprechen. C soll auf T gesetzt werden, wenn die Addition auf einen Übertrag in der Stelle höchster Ordnung führt. Auf diese Weise haben wir einen elektrischen Schaltkreis symbolisiert, der als Halbadder bezeichnet wird.

12) Ändere das Programm zu Aufgabe 11) so ab, daß sowohl bei der Eingabe als auch bei der Ausgabe die Prüfung auf ungerade Parität berücksichtigt wird, d.h. die Werte von A(5) bzw. B(5) sind einzulesen bzw. auszugeben.

16.3. Funktionsdefinitionen (Funktionsvereinbarungen)

Durch einzelne Statements können in einem FORTRAN-Programm Funktionen definiert werden. Eine solche Funktionsvereinbarung besitzt die Form:

name($p_1, \cdots, p_n$)=ausdruck

Hierbei bedeuten die p_i die Parameter der Funktion und unter ausdruck kann jeder beliebige arithmetische oder auch logische Ausdruck geschrieben werden. Die Anzahl der Parameter darf 15 nicht übersteigen. Unter name kann jeder gültige Variablennamen niedergeschrieben werden; die Parameter selbst werden durch nichtindizierte Variablennamen angezeigt. Jeder Parameter gehört genau so wie die regulären Variablen zum Typ INTEGER*4 bzw. REAL*4; es sei denn, daß der Typ explizit oder implizit (durch das IMPLICIT-Statement) anders festgelegt wird.

Die Funktion muß durch ein einziges Statement definiert werden. Die für die Parameter verwendeten Namen haben nur für dieses eine Statement eine Bedeutung; sie bezeichnen quasi Scheinvariable. Zwischen ihnen und evtl. im Rest des Programmes auftretenden gleichen Namen besteht keinerlei Beziehung. Der auf der rechten Seite der Funktionsdefinition stehende Ausdruck darf beliebige Konstanten und Variablen enthalten; es braucht nur dafür Sorge getragen zu werden, daß allen Variablen Werte zugewiesen sind, wenn die Funktion aufgerufen wird, d.h. wenn sie berechnet wird.

Verfolgen wir einige Beispiele von Funktionsvereinbarungen:

```
ERSTF(Y) = 3.Ø*Y + 7.4
ZWEIF(Y) = 8.Ø**Y + AFR
DREIF(X,Y) = SQRT(X*X+Y*Y)
```

Wenn der Funktionswert von ZWEIF(Y) berechnet werden soll, so muß zuvor AFR definiert sein, d.h. dieser Variablen muß zuvor ein Wert zugewiesen worden sein. Funktionsvereinbarungen müssen in einem Programm vor allen Anweisungen erscheinen. Man kann sie in einem Programm so aufrufen wie jede Funktion, die in die Bibliothek aufgenommen ist (z.B. SIN).

Das nachfolgende Programm (siehe Abb. 16.6.) verwendet zwei Funktionsdefinitionen. Diesem Programm liegt folgende Aufgabenstellung zugrunde: Es liest Lochkarten ein,

auf denen die Werte eines Winkels und zweier Seiten eines Dreiecks abgelocht sind, und es druckt alle Seiten und Winkel sowie den Flächeninhalt des betreffenden Dreiecks aus. Für die Seiten haben wir die Namen SA, SB und SC gewählt, für die Winkel die Namen A, B und C und für den Flächeninhalt den Namen FLIN. Wir wollen alle Winkel im Gradmaß angeben, obgleich die in die FORTRAN-Sprache aufgenommenen Funktionen sich ausschließlich des Bogenmaßes bedienen. Um die einzelenen zu berechnenden Dreiecke unterscheiden zu können, verwenden wir für jedes Dreieck eine Kennzeichnung, die wir im Programm mit T bezeichnet haben. Für die Kennzeichnung haben wir einen vierstelligen Zeichenschlüssel vorgesehen. Die FORMAT-Vereinbarungen für die Ausgabe sind für den Drucker codiert, infolgedessen enthalten sie auf ihrer ersten Stelle das Steuerzeichen, das für die Steuerung des Druckermechanismus benötigt wird (Vorschub bzw. Zeilentransport). Ergeben sich bei der Berechnung eines Dreiecks zwei Lösungen, so wird die zweite unter die erste gedruckt; wir vermeiden jedoch in einem solchen Fall das erneute Drucken der Eingabewerte.

```
      SIND(A)=SIN(Ø.Ø1745329*A)
      ASIND(Y)=57.29578*ATAN(Y/SQRT(1.Ø-Y*Y))
      WRITE (3,3Ø1)
    1 READ (1,1Ø1)  T,A,SA,SB
      G=SB*SIND(A)/SA
      IF  (G-1.Ø)  5,5,3
    5 B=ASIND(G)
      C=18Ø.Ø-A-B
      SC=SA*SIND(C)/SIND(A)
      FLIN=Ø.5*SB*SC*SIND(A)
      WRITE (3,3Ø2)  T,A,B,C,SA,SB,SC,FLIN
      B=18Ø.Ø-B
      C=18Ø.Ø-A-B
      IF  (C)  1,1,2
    2 SC=SA*SIND(C)/SIND(A)
      FLIN=Ø.5*SB*SC*SIND(A)
      WRITE (3,3Ø3)  B,C,SC,FLIN
      GO TO 1
    3 WRITE (3,3Ø4)  T
      GO TO 1
  1Ø1 FORMAT (A4,F1Ø.3,2E14.6)
  3Ø1 FORMAT ('1','DR.',5X,'A',9X,'B',9X,'C',
     1        9X,'SEITE A',7X,'SEITE B',7X,'SEITE C',
     2        8X,'FLIN')
  3Ø2 FORMAT (/'Ø',A4,3F1Ø.3,4E14.6)
  3Ø3 FORMAT (1X,14X,2F1Ø.3,28X,2E14.6)
  3Ø4 FORMAT ('Ø',A4,5X,'KEINE LOESUNG')
      END
```

Abb. 16.6. Dreiecksberechnungen

Es sollte noch vermerkt werden, daß A und Y in den Funktionsvereinbarungen die Namen von Parametern darstellen; zwischen dem Parameter A und dem Variablennamen A im Programm besteht deshalb keine Verbindung.

Die Verwendung von logischen Variablen und logischen Ausdrücken bei Funktionsdefinitionen ist selbstverständlich nur auf die volle FORTRAN-Sprache beschränkt.

16.4. Funktionsunterprogramme

Die Unterprogramme von FUNCTION-Typ unterscheiden sich in zweierlei Hinsicht von den Funktionsvereinbarungen:

a) Die Definition der Funktion braucht sich nicht auf die Verwendung eines einzigen Statements zu beschränken.

b) Ein Funktionsunterprogramm kann zusammen mit dem Hauptprogramm, aber auch getrennt von ihm, übersetzt werden.

Ein Funktionsunterprogramm besitzt folgenden allgemeinen Aufbau:

```
FUNCTION name(p1,···,pn)
:
name = ausdruck
:
RETURN
:
END
```

Ein Funktionsunterprogramm muß also mit der Vereinbarung FUNCTION beginnen. Die Namen von Funktionen können nach den gleichen Aspekten ausgewählt werden wie die Namen von normalen Variablen, d.h. ein mit den Buchstaben I, J, K, L, M oder N beginnender Namen kennzeichnet eine Funktion, deren Funktionswert ganzzahlig ist, ein Name, der mit einem anderen Buchstaben beginnt, eine Funktion, deren Wert reell ist. Will man von diesen implizit festgelegten Typzuordnungen abweichen, so muß man die FUNCTION-Vereinbarung in folgender Form codieren:

```
typ FUNCTION name(p1,···,pn)
```

Unter typ ist hierbei entweder REAL oder INTEGER oder DOUBLE PRECISION einzusetzen.

In der vollen FORTRAN-Sprache kann die explizite Typfestlegung für eine Funktion auch in der folgenden Form vorgenommen werden:

```
typ FUNCTION name s(p1,···,pn)
```

Unter typ kann hierbei REAL oder INTEGER oder COMPLEX oder LOGICAL angegeben werden; die zusätzliche Angabe einer Länge s ist freigestellt.

Für jede Funktion muß mindestens ein Parameter angegeben sein. Als Parameter können nichtindizierte Variable, Bereiche und die Namen von Funktionen bzw. anderen Unterprogrammen gewählt werden. Ein Funktionsunterprogramm kann sich aus einer beliebigen Anzahl von Statements zusammensetzen, doch das letzte Statement, das in einem Unterprogramm ausgeführt werden soll, muß unbedingt die RETURN-Anweisung sein. Diese Anweisung übergibt die Programmablaufsteuerung zurück an das Hauptprogramm; sie stellt also die logisch letzte Anweisung eines Programmes dar. Das END-Statement muß dagegen als physisch letztes Statement eines Funktionsunterprogrammes auftreten. In jedem Funktionsunterprogramm können mehrere RETURN-Anweisungen erscheinen, jedoch nur ein END-Statement. Funktionen, die durch ein FUNCTION-Statement definiert werden, können genau so aufgerufen werden wie die Funktionen, deren Definition durch eine einzige Funktionsvereinbarung vorgenommen werden kann.

Als Beispiel wollen wir ein Programm betrachten, das zur Berechnung von

$$P(m,n,r) = C(n,m) \cdot C(r-1, n-m-1) : C(n+r-1, r)$$

für $m=0, 1, \cdots, n$ dienen soll. Der Wert der Funktion C(x, y) gibt die Anzahl der Kombinationen von y-Elementen an, die man sich aus einer Menge von x-Elementen herausgreift. C(52, 13) gibt z. B. an, auf wieviel verschiedene Weisen 13 Karten aus einem aus 52 Karten bestehenden Kartenspiel einem Mitspieler ausgehändigt werden können. Der Funktionswert P(m, n, r) spiegelt die Wahrscheinlichkeit wider, daß m Zellen leer bleiben, wenn r nicht voneinander unterscheidbare Dinge auf n Zellen verteilt werden und jede unterscheidbare Anordnung gleichwahrscheinlich ist. Man sagt in der Physik, daß Teilchen, die diese Eigenschaft besitzen, dem statistischen Gesetz von Bose-Einstein gehorchen.

In unserem Programm wollen wir uns auf zwei Funktionsunterprogramme beziehen können; in dem einen soll die Anzahl der Kombinationen berechnet werden, d. h. die Bestimmung von

$$C(x,y) = \frac{x!}{y!\ (x-y)!}$$

erfolgen, in dem anderen soll die Berechnung von Fakultäten vorgenommen werden.

Zur Berechnung der Fakultäten soll die bekannte Definition derselben herangezogen werden:

$$0! = 1$$
$$x! = x \cdot (x-1)!$$
$$= x \cdot (x-1) \cdot (x-2) \cdots 3 \cdot 2 \cdot 1$$

Wenden wir jetzt unsere Aufmerksamkeit den einzelnen Programmeinheiten zu, die nacheinander in den Abb. 16.7. bis 16.9. aufgeführt sind.

```
C       HAUPTPROGRAMM
C       ==============
      1 READ (1,1Ø1)  N,IR
        WRITE (3,3Ø1) N,IR
        IF  (N)  2,2,3
      2 STOP
      3 IF  (IR)  2,2,4
      4 M=Ø
      5 P=COMBF(N,M)*COMBF(IR-1,N-M-1) / COMBF(N+IR-1,IR)
        WRITE (3,3Ø2)  M,P
        M=M+1
        IF  (M-N)  5,5,1
   1Ø1  FORMAT (2I8)
   3Ø1  FORMAT (///' N=',I5,5X,'R=',I5//6X,'M',5X,'WSCH.')
   3Ø2  FORMAT (I8,F14.1Ø)
        END
```

Abb. 16.7. Hauptprogramm

Man bemerkt hierbei, daß jedes Unterprogramm praktisch ein gesondertes Programm darstellt. Wir wollen deshalb besser folgende Terminologie einführen: Unser Programm besteht aus drei Programmeinheiten, dem Hauptprogramm und zwei Funktionsunterprogrammen. Die Variablen in den Unterprogrammen, denen Werte von der aufrufenden Programmeinheit übergeben werden, wollen wir, wie schon eingeführt, als Parameter bezeichnen. Die Statementnummern eines Unterprogrammes sind lokal, d. h. sie besitzen nur für das Unterprogramm selbst Gültigkeit; das gleiche gilt für die Namen der Variablen des Unterprogrammes, gleichgültig, ob diese Parameter sind oder nicht.

```
C       FUNKTIONSUNTERPROGRAMM 1
C       ZUR BERECHNUNG VON FAKULTAETEN
C       ==============================
        FUNCTION  FAKT(N)
        IF  (N)  11,12,13
   11   FAKT = Ø.Ø
        RETURN
   12   FAKT = 1.Ø
        RETURN
   13   X = 1.Ø
        DO 14  I=1,N
   14   X = X*I
        FAKT = X
        RETURN
        END
```

Abb. 16.8. Funktionsunterprogramm zur Berechnung von Fakultäten

```
C       FUNKTIONSUNTERPROGRAMM 2
C       ZUR BERECHNUNG VON KOMBINATIONEN
C       ================================
        FUNCTION  COMBF(N,K)
        D = FAKT(N-K) * FAKT(K)
        IF  (D)  11,11,12
   11   COMBF = Ø.Ø
        RETURN
   12   COMBF = FAKT(N) / D
        RETURN
        END
```

Abb. 16.9. Funktionsunterprogramm zur Berechnung von Kombinationen

Parameter dürfen in einer EQUIVALENCE-Vereinbarung nicht aufgeführt werden. Als Parameter darf auch ein Bereich verwendet werden. In einem solchen Fall muß das Argument, d.h. die Größe, die beim Aufruf eines Unterprogrammes an die Stelle des entsprechenden Parameters tritt, ebenfalls ein Bereich sein. Weiterhin muß der als Parameter auftretende Bereich in einer DIMENSION-Vereinbarung des Unterprogrammes erscheinen, der als Argument verwendete Bereich dagegen in einer DIMENSION-Vereinbarung der aufrufenden Programmeinheit. Dabei müssen die Dimensionen, die im Unterprogramm für einen Bereichsparameter festgelegt werden, mit den Dimensionen übereinstimmen, die in der aufrufenden Programmeinheit für das Bereichsargument festgesetzt werden.

In der vollen FORTRAN-Sprache ist es bei Unterprogrammen erlaubt, variable Dimensionsgrenzen einzusetzen, d.h. in der DIMENSION-Verei nbarung Variable des ganzzahligen Typs zur Kennzeichnung der Indizes zu verwenden. Das gilt aber wohlgemerkt nicht für Hauptprogramme. Betrachten wir dazu das Beispiel in Abb. 16.10.

Immer, wenn diese sogenannten einstellbaren Dimensionen in einem Unterprogramm verwendet werden, müssen deren Werte durch Argumente in der aufrufenden Programmeinheit festgelegt werden. Die Möglichkeit, einstellbare Dimensionen zu benutzen, erleichtert wesentlich den Einsatz eines Programmes , das Bereiche unterschiedlichen Umfanges ansprechen muß.

Um ein aus mehreren Programmeinheiten bestehendes Programm zu übersetzen und ablaufen zu lassen, müssen wir für die einzelnen Betriebssysteme gewisse Vorkehrungen treffen, d.h. bestimmte Steuerkarten präparieren und ein Kartenpaket zusammen-

```
Hauptprogramm:   DIMENSION  A(5,3)
                 J = 5
                 A = 7.46
                 X = BK(U,A,J,3)
                 :
Unterprogramm:   FUNCTION  BK(X,A,I,L)
                 DIMENSION  A(I,L)
                 :
```

Abb. 16.10. Variable Dimensionsgrenzen in Unterprogrammen

stellen, das der Aufgabenstellung gerecht wird. Für das Betriebssystem DOS sieht dieses Kartenpaket (zugeschnitten auf das in den Abb. 16.7. bis 16.9. dargestellte Beispiel) so aus, wie es in der Abb. 16.11. dargestellt ist.

```
// JOB BEISPI
---Falls erforderlich: Zuordnungskarten(ASSGN)---
// OPTION LINK
PHASE BEISPI,*
// EXEC FFORTRAN
(Hauptprogramm)
/*
// EXEC FFORTRAN
(Funktionsunterprogramm 1)
/*
// EXEC FFORTRAN
(Funktionsunterprogramm 2)
/*
// EXEC LNKEDT
// EXEC
(Datenkarten)
/*
/&
```

Abb. 16.11. Steuerstatements zum Übersetzen, Binden und Ausführen des in den Abb. 16.7. bis 16.9. vorgestellten Programmkomplexes (Betriebssystem DOS)

Die Reihenfolge, in der die drei Programmeinheiten übersetzt werden, ist unwesentlich. Sollen Unterprogramme aus der Programmbibliothek ebenfalls in das Objektprogramm eingefügt werden, dann müssen wir vor die Karte

```
// EXEC LNKEDT
```

noch INCLUDE-Karten legen, z. B.

```
INCLUDE SIN
INCLUDE SQRT
```

usw. Wird das entsprechende Unterprogramm implizit im Programm aufgerufen, so können die INCLUDE-Karten weggelassen werden.

Wenden wir uns nun dem Betriebssystem OS zu und schauen uns einmal hier die Zusammenstellung der in der Jobsteuersprache [1] abgefaßten Steuerstatements an (Abb. 16.12.).

Das Hauptprogramm ist als erste Programmeinheit zu übersetzen. Die Funktionsunterprogramme schließen sich daran an, wobei die Reihenfolge ihrer Übersetzung keine Rolle spielt.

1) Im angloamerikanischen Sprachraum als Job Control Language, abgekürzt JCL bezeichnet.

```
//BEISPI  JOB  ...
//HAUPT   EXEC  FORTGC
//FORT.SYSIN DD *
(Hauptprogramm)
//FUP1    EXEC  FORTGC
//FORT.SYSIN DD *
(Funktionsunterprogramm 1)
//FUP2    EXEC  FORTGCLG
//FORT.SYSIN DD *
(Funktionsunterprogramm 2)
//GO.SYSIN DD *
(Datenkarten)
/*
```

Abb. 16.12. Steuerstatements zum Übersetzen, Binden und Ausführen des in der Abb. 16.7. bis 16.9. vorgestellten Programmkomplexes (Betriebssystem OS)

Aufgabengruppe 43

1) Der Gegenwartswert S einer Kapitalanlage von 1.- DM ist gegeben durch

$$S = \frac{(1+r)^n - 1}{r}$$

Hierbei bedeutet r den Zinsfuß und n ist gleich der Zahl der abgelaufenen Zinsperioden. Eines der Probleme bei Geldanlagen besteht aber gerade darin, den Zinsfuß zu bestimmen, wenn S und n bekannt sind. Bei der Berechnung von r kann man so vorgehen, daß man für r zunächst zwei Näherungswerte, r_1 und r_2, so aufsucht, daß $S_1 < S < S_2$ ist. S_1 bzw. S_2 findet man, wenn man r_1 bzw. r_2 in die Gleichung für S einsetzt.

Nach Ermittlung von r_1 und r_2 bestimmt man einen neuen Näherungswert für r durch Einsetzen der Werte von S, S_1, S_2, r_1, r_2 in die Interpolationsgleichung

$$\frac{S-S_1}{S_2-S_1} = \frac{r_i-r_1}{r_2-r_1}$$

Unter r_i ergibt sich hierbei der neue Näherungswert. Danach rechnet man S_i durch Einsetzen von r_i in die für S angegebene Formel aus und ersetzt S_1 oder S_2 durch S_i, je nachdem, ob $S_i < S$ oder ob $S_i > S$ ist. Nach diesem Austausch wird ein neuer Näherungswert für r_i errechnet und daraus S_i bestimmt usw. Dieses Verfahren wird solange fortgesetzt, bis ein Wert für r_i gefunden ist, dessen zugehöriger Wert S_i genügend nahe bei S liegt.

Schreibe ein Programm, mit dessen Hilfe r bis auf sechs Dezimalstellen genau berechnet werden kann. Zur Eingabe gelangen die Werte von n und S; ausgegeben werden sollen die Werte von n, S und r.

Zur Berechnung von S_i in Abhängigkeit von r_i soll im Programm eine arithmetische Funktion definiert werden.

2) Die kumulative binomische Wahrscheinlichkeit ist durch die Formel

$$B(k; n, p) = \sum_{r=0}^{k} C(n, r)\, p^r\, (1-p)^{n-r}$$

gegeben.

Schreibe ein Programm zur Berechnung von B in Abhängigkeit von k. Zur Eingabe gelagen Werte für n und für p. Es soll eine Tabelle der Wahrscheinlichkeitswerte für k=0, 1, 2, ···, n gedruckt werden.

Zur Berechnung der Werte der kombinatorischen Funktion C(n, r) soll ein Funktionsunterprogramm verwendet werden.

3) Schreibe ein Programm zur Berechnung der Reihe, deren einzelne Glieder a_i dem Bildungsgesetz

$$a_i = \frac{1\ (1+c)\ (1+2c)\ (1+3c)\ \cdots\ (1+ic}{2\ (2+c)\ (2+2c)\ (2+3c)\ \cdots\ (2+ic)i} \qquad \text{für } i=1,\ 2,\ \cdots$$

genügen.
Die Ergebnisse sind in einer zweispaltigen Tabelle auszugeben. In der linken Spalte sind die Werte von c, in der rechten Spalte die Werte von $\sum_{i=1}^{\infty} a_i$ zu drucken. Für c sind nacheinander 1, 1/2, 2, 1/3, 3, 1/4, 4, ··· einzusetzen.

Man beachte hierbei, daß die Reihe nicht für alle Werte von c konvergiert.

4) Zur näherungsweisen Berechnung von n! können die beiden nachfolgenden nach Stirling benannten Formeln dienen:

a) $\sqrt{2\pi n}\cdot\left(\frac{n}{e}\right)^n$

b) $\sqrt{2\pi n}\cdot\left(\frac{n}{e}\right)^n\ e^{\frac{1}{4n}}$

Berechne in einem Programm Näherungswerte für n! nach beiden Formeln. Dividiere beide Näherungswerte durch n!, um zu zeigen, welche der beiden genauere Resultate liefert. Es soll eine Tabelle gedruckt werden, die neben den Werten von n und n! auch die Werte dieser beiden Verhältnisse enthält. Zur Ausgabe der Werte von n! soll die E-Formatangabe, für die anderen drei Werte die F-Formatangabe verwendet werden.

5) Man betrachte das unendliche Produkt

$$P = \prod_{n=2}^{\infty} \frac{n}{z} \cdot \sin\left(\frac{z}{n}\right)$$

Nach Definition ist P=1, wenn z=0 ist.

Schreibe ein Programm zur Berechnung von P. Die Werte von z werden über Karten eingelesen (ein Wert pro Karte). P konvergiert für alle Werte von z, so daß es zu keinem Überlauf kommen kann.

16.5. Unterprogramme vom SUBROUTINE-Typ (Subroutine-Unterprogramme)

Unterprogramme vom FUNCTION-Typ (Funktionsunterprogramme) zeichnen sich dadurch aus, daß sie wohl von vielen Parametern abhängig gemacht werden können, daß mit ihrer Hilfe aber stets nur ein einziges Ergebnis bestimmt werden kann, nämlich der Funktionswert. Unterprogramme vom SUBROUTINE-Typ (Subroutine-Unterprogramme) können dagegen zur Ermittlung einer beliebigen Anzahl von Resultaten verwendet werden. Subroutine-Unterprogramme sind wie folgt aufgebaut:

```
SUBROUTINE name(p1,···,pn)
    ⋮
RETURN
END
```

Sie werden also durch die SUBROUTINE-Vereinbarung eingeleitet. Wie bei Funktionsunterprogrammen muß die RETURN-Anweisung die letzte logische Anweisung einer Programmeinheit sein, die ein Subroutine-Unterprogramm darstellt. Durch die RETURN-Anweisung wird die Programmablaufsteuerung wieder der aufrufenden Programmeinheit übergeben. Die END-Vereinbarung muß wie üblich als letztes physisches Statement des Unterprogrammes erscheinen. Wie die Funktionsunterprogramme stellen auch die Subroutine-Unterprogramme eigenständige Programme dar, die vom Hauptprogramm getrennt sind, d.h. sie sind Programmeinheiten. Alle Variablen, die im Unterprogramm verwendet werden, sind damit nur für das Unterprogramm definiert, d.h. es sind lokale Namen. Das gleiche gilt auch für die im Unterprogramm benutzen Statementnummern. Variablennamen wie auch Statementnummern haben somit nichts Gemeinsames mit gleichen Variablennamen und gleichen Statementnummern der aufrufenden Programmeinheit.

Subroutine-Unterprogramme können mit beliebigen Namen benannt werden, doch müssen diese im Sinne der FORTRAN-Sprache gültige Namen sein. Für diese Namen ist keine Typvereinbarung erforderlich, da in einem Subroutine-Unterprogramm, wie schon gesagt, mehrere Ergebnisse errechnet werden können (es muß jedoch nicht unbedingt ein solches überhaupt gebildet werden). Innerhalb eines Subroutine-Unterprogrammes dürfen mit Ausnahme der Statements FUNCTION, SUBROUTINE und BLOCK DATA (siehe Abschnitt 16.7.) alle anderen Statements verwendet werden. Im Gegensatz zu den Funktionsunterprogrammen brauchen Subroutine-Unterprogramme nicht unbedingt von Parametern abhängig gemacht zu werden; auch darf der Name des Unterprogrammes selbst nicht im Unterprogramm als Variablennamen auftreten, wie das bei Funktionsunterprogrammen sogar der Fall sein muß. Im übrigen gelten für die Parameter die gleichen Gesetzmäßigkeiten wie für die Parameter von Funktionsunterprogrammen.

Explizit führt ein Subroutine-Unterprogramm niemals auf ein bestimmtes Resultat. Implizit dürfen durch ein Subroutine-Unterprogramm jedoch beliebig viele Ergebnisse ermittelt werden. Betrachten wir hierzu die Statementfolge von Abb. 16.13.

```
SUBROUTINE  PUPPE(X,Y,Z,W)
Z = X + Y
W = X - Y
RETURN
END
```

Abb. 16.13. Beispiel für ein Unterprogramm des SUBROUTINE-Typs

Wenn dieses Unterprogramm aufgerufen wird, so erwartet es, daß ihm für die beiden Parameter X und Y Werte übergeben werden; bei der Rückkehr zur aufrufenden Programmeinheit stellt es dieser selbst Werte unter den Parametern Z und W zur Verfügung. Um Unterprogramme vom SUBROUTINE-Typ aufrufen zu können, müssen wir das CALL-Statement einsetzen. Dieses Statement besitzt die Form:

```
CALL name(a1,···,an)
```

Hierbei ist unter name der Name des aufzurufenden Unterprogrammes zu verstehen. Um das oben aufgeführte Subroutine-Unterprogramm in ein Programm einzufügen, d.h. aufzurufen, könnten wir z.B. die folgenden beiden Statements verwenden:

```
X = 3.4
CALL PUPPE(7.Ø,X,Y,P)
```

Bei Rückkehr vom Unterprogramm haben damit Y den Wert 10.4 und P den Wert 3.6 zugewiesen erhalten. Voraussetzung hierfür ist selbstverständlich, daß das Unterprogramm PUPPE auch existiert.

Um die Verwendung dieser Statements noch eingehender erklären zu können, wollen wir ein Programm (Abb. 16.14.) betrachten, das das folgende Problem löst: Es sollen die Koeffizienten der quadratischen Gleichung $ax^2 + bx + c = Ø$ eingelesen und die Lösungen (Wurzeln) der betreffenden Gleichung ausgegeben werden. Alle Koeffizienten sind komplexe Zahlen. Bei Benutzung der vollen FORTRAN-Sprache würden wir normalerweise die Variablen, denen die Koeffizienten als Werte zugewiesen werden sollen, als Variable des komplexen Typs definieren, doch in unserem Falle wollen wir das Problem in der BASIC FORTRAN-Sprache lösen. Wir benötigen deshalb Subroutine-Unterprogramme zur Durchführung der Multiplikation, der Division und der Ermittlung von Quadratwurzeln mit komplexen Operanden (Abb. 16.15. bis 16.17.).

```
C       HAUPTPROGRAMM
C       ============
        DIMENSION  A(2),B(2),C(2),
       1           D(2),E(2),F(2),R(4)
   2Ø   READ (1,1Ø1)  A,B,C
        CALL CMUL(B,B,D)
        E(1) = 4.Ø*A(1)
        E(2) = 4.Ø*A(2)
        CALL CMUL(E,C,F)
        E(1) = D(1) - F(1)
        E(2) = D(2) - F(2)
        CALL CSQR(E,F)
        E(1) = -2.Ø*A(1)
        E(2) = -2.Ø*A(2)
        CALL CDIV(F,E,D)
        CALL CDIV(B,E,F)
        R(1) = F(1) + D(1)
        R(2) = F(2) + D(2)
        R(3) = F(1) - D(1)
        R(4) = F(2) - D(2)
        WRITE (3,3Ø1)  R
        GO TO 2Ø
  1Ø1   FORMAT (6E12.6)
  3Ø1   FORMAT (4E14.6)
        END
```

Abb. 16.14. Hauptprogramm zur Berechnung von Lösungen quadratischer Gleichungen mit komplexen Koeffizienten

Es wäre noch zu bemerken, daß ein Programmierer sich nicht mit Einzelheiten über die Arbeitsweise von Unterprogrammen beschäftigen muß, wenn in einer Installation nur standardisierte Unterprogramme verwendet werden. Diese könnten dann in jedem FORTRAN-Programm aufgerufen werden.

```
C       SUBROUTINE-UNTERPROGRAMM
C       ========================
        SUBROUTINE  CMUL(A,B,C)
        DIMENSION  A(2),B(2),C(2)
        C(1) = A(1)*B(1) - A(2)*B(2)
        C(2) = A(1)*B(2) + A(2)*B(1)
        RETURN
        END
```

Abb. 16.15. Subroutine – Unterprogramm zur Bestimmung des Produktes zweier komplexer Zahlen

```
C       SUBROUTINE-UNTERPROGRAMM
C       ========================
        SUBROUTINE  CDIV(A,B,C)
        DIMENSION  A(2),B(2),C(2)
        D = B(1)*B(1) + B(2)*B(2)
        C(1) = (A(1)*B(1)+A(2)*B(2)) / D
        C(2) = (A(2)*B(1)-A(1)*B(2)) / D
        RETURN
        END
```

Abb. 16.16. Subroutine – Unterprogramm zur Bestimmung des Quotienten zweier komplexer Zahlen

```
C       SUBROUTINE-UNTERPROGRAMM
C       ========================
        SUBROUTINE  CSQR(A,C)
        DIMENSION  A(2),C(2)
        D = (A(1)**2 + A(2)**2) ** Ø.25
        E = Ø.Ø
        IF  (A(1))  61,62,63
   62   E = Ø.7853982
        IF  (A(2))  64,65,65
   64   E = -Ø.7853982
        GO TO 65
   61   E = 1.57Ø796
   63   E = E + Ø.5*ATAN(A(2)/A(1))
   65   C(1) = D*COS(E)
        C(2) = D*SIN(E)
        RETURN
        END
```

Abb. 16.17. Subroutine – Unterprogramm zur Bestimmung der Quadratwurzel aus einer komplexen Zahl

Muß man ein größeres Programm anfertigen, so wird man zweckmäßigerweise nur ein kleines Hauptprogramm (gewissermaßen als Rahmenprogramm) schreiben, das ein Unterprogramm nach dem anderen aufruft. Bei einem solchen Vorgehen kann der Programmierer jedes Unterprogramm getrennt testen; damit kann er die Anstrengungen und die Aufwendungen, die zur Fehlersuche und zur Beseitigung von Irrtümern erforderlich sind, erheblich reduzieren. Diese Methode gestattet es auch, erforderlich Programmänderungen bzw. -verbesserungen schneller und einfacher durchzuführen. Bei Programmen mit über tausend FORTRAN-Statements ist allein der Unterschied der für die Bereinigung von Programmirrtümern erforderlichen Zeiten schon in beachtlichem Maße zu verspüren.

Wir haben bereits erwähnt, daß als Argumente beim Aufruf von Unterprogrammen des FUNCTION- bzw. des SUBROUTINE-Typs auch die Namen von Funktions- oder Subrou-

tine-Unterprogrammen verwendet werden dürfen. Verfolgen wir dazu ein Beispiel und betrachten deshalb das Unterprogramm:

```
SUBROUTINE  CLARA(A,B,C)
C = B(A)
RETURN
END
```

Die aufrufende Programmeinheit möge folgende Statements enthalten:

```
    .
    .
CALL  CLARA(7.45+2 X,SIN,Y)
    .
    .
CALL  CLARA(RU,COS,YET)
    .
    .
```

Durch die erste CALL-Anweisung wird erreicht, daß der Variablen Y der Wert von SIN(7.45+2*X) zugewiesen wird; durch die zweite CALL-Anweisung wird Z auf den Wert COS(RU) gesetzt. Der Name irgendeines Unterprogrammes kann bei einem Aufruf als Argument angegeben werden. Es ist deshalb unbedingt notwendig, daß in das Hauptprogramm eine EXTERNAL-Vereinbarung aufgenommen wird, damit die als Argumente vorgesehenen Unterprogrammnamen festgehalten werden. Für den obigen Fall müßte diese wie folgt lauten:

```
EXTERNAL  SIN,COS
```

Durch die EXTERNAL-Vereinbarung wird also festgelegt, welche Unterprogrammnamen als Argumente beim Aufruf anderer Unterprogramme Verwendung finden. Die allgemeine Form des EXTERNAL-Statements sieht wie folgt aus:

```
EXTERNAL  name1,name2,···
```

Tritt in einem CALL-Statement der Name eines Unterprogrammes auf, so wird der Name selbst dem Unterprogramm übergeben; in allen anderen Fällen wird die Hauptspeicheradresse der Variablen, der Konstanten oder des Ausdruckes übertragen.

In der vollen FORTRAN-Sprache ist es möglich, einem aufgerufenen Unterprogramm den Namen des Argumentes und nicht dessen Wert zu übergeben, d.h. der Namensaufruf ist nicht auf die Namen von Unterprogrammen beschränkt. Wenn es zu einem Namensaufruf kommen soll, muß das bei der Programmierung des Unterprogrammes bereits festgelegt werden. Der Parameter, für den der Namensaufruf eintreten soll, muß im SUBROUTINE-Statement durch Schrägstriche eingeschlossen werden. Beispielsweise werden beim Aufruf des Unterprogrammes, das durch die Vereinbarung

```
SUBROUTINE  BUL(A,/B/,C)
```

eingeleitet wird, dem Unterprogramm die augenblicklichen Werte des ersten und des dritten Argumentes sowie der Name des zweiten Argumentes übergeben. Grundsätzlich wäre hierzu noch zu bemerken, daß beim Aufruf eines Unterprogrammes diesem die Namen von Bereichen und von Unterprogrammen auch dann übergeben werden, wenn der ihnen entsprechende Parameter nicht in Schrägstriche eingeschlossen ist; für alle anderen Argumente kommt es nur dann zu einer Namensübertragung, wenn der ihnen entsprechende Parameter in Schrägstriche eingeschlossen ist.

Wenn ein Unterprogramm aufgerufen wird, so beginnt der Ablauf desselben normalerweise bei seiner ersten Anweisung. Der Logik des Unterprogrammes entsprechend wer-

den dann die anderen Anweisungen nacheinander solange ausgeführt, bis die Programmablaufsteuerung auf eine RETURN-Anweisung stößt; wird eine solche festgestellt, so erfolgt eine Rückkehr zu der Anweisung der aufrufenden Programmeinheit, die unmittelbar auf das CALL-Statement folgt. Bei der vollen FORTRAN-Sprache kann diese Norm in zweierlei Hinsicht modifiziert werden.

Die ENTRY-Vereinbarung wird dann verwendet, wenn die Ausführung des aufgerufenen Unterprogrammes mit einer anderen Anweisung als der ersten beginnen soll. Das ENTRY-Statement besitzt folgende allgemeine Form:

ENTRY name(p_1,···,p_n)

Dieses Statement kann auch bei Funktionsunterprogrammen eingesetzt werden, es ist also nicht auf die Verwendung bei Subroutine-Unterprogrammen beschränkt. Der dem Schlüsselwort ENTRY folgende Eingangsname [1] und die anschließende Parameterliste unterliegen denselben Regeln wie die Namen und Parameterlisten bei den FUNCTION- und SUBROUTINE-Vereinbarungen. Wird ein Unterprogramm dadurch aufgerufen, daß man sich auf einen Eingangsnamen desselben und nicht auf seinen regulären Namen bezieht, so wird die Programmablaufsteuerung von der aufrufenden Programmeinheit derjenigen Anweisung des Unterprogrammes übergeben, die unmittelbar auf die ENTRY-Vereinbarung folgt. Wir wollen das an einem Beispiel demonstrieren (Abb. 16.18.).

```
      SUBROUTINE  PECR(X,Y,P)
      IF  (Y)  11,11,12
11    X = -X
12    Y = X + Y
      ENTRY  PECT(Y,P)
      P = 5.Ø * Y**Ø.25
      RETURN
      END
```

Abb. 16.18. Unterprogramm mit ENTRY-Statement

In der aufrufenden Programmeinheit mögen folgende Statements codiert sein (Abb. 16.19.).

```
      .
      .
      .
X = 78.Ø
Y = 3.Ø
CALL PECR(X,Y,Z)
CALL PECT(4.Ø,P)
      .
      .
      .
```

Abb. 16.19. Aufruf eines Unterprogrammes

Durch die erste CALL-Anweisung wird Z der Wert 15.0 zugewiesen; durch die zweite CALL-Anweisung erhält P den Wert $0.5\sqrt{2}\approx 0.7071068$. Es ist selbstverständlich erforderlich, daß die Parameterliste mit der Argumentenliste in der CALL-Anweisung übereinstimmt, soweit es die Anzahl, die Reihenfolge und den Typ der Argumente bzw. der Parameter betrifft. Zwischen der Parameterliste eines ENTRY-Statements und der Parameterliste des zugehörigen FUNCTION- oder SUBROUTINE-Statements braucht dagegen natürlich keine Übereinstimmung zu herrschen.

1) Ein Eingangsname muß sich selbstverständlich von dem Namen unterscheiden, der in der SUBROUTINE- bzw. FUNCTION-Vereinbarung als Unterprogrammnamen festgelegt ist.

Die andere für die volle FORTRAN-Sprache zur Verfügung stehende Möglichkeit basiert auf einer abgewandelten Form der RETURN-Anweisung; diese lautet:

```
RETURN i
```

Hierbei ist i eine ganzzahlige Konstante oder eine ganzzahlige Variable. Normalerweise erfolgt bekanntlich die Rückkehr vom Unterprogramm zu der Anweisung der aufrufenden Programmeinheit, die unmittelbar hinter der Anweisung steht, die das Unterprogramm aufgerufen hat. Durch Einfügung von einem oder von mehreren Sternzeichen in die Parameterliste des SUBROUTINE-Statements, können wir als Argumente in der CALL-Anweisung auch Statementnummern benutzen. Diesen müssen allerdings &-Zeichen vorausgehen, damit es zu keinen Verwechslungen mit Konstanten kommen kann. Wird nun die Anweisung

```
RETURN i
```

ausgeführt. so wird bei der Rückkehr in die aufrufende Programmeinheit die Programmablaufsteuerung der Anweisung mit der Statementnummer übergeben, die als i-te Statementnummer inmitten der Argumentenliste steht. Verfolgen wir zur näheren Erläuterung ein Beispiel:

```
      SUBROUTINE  RWY(Z,C,*,*,W,*)
      IF  (Z)  1,2,2
    1 W = 4.Ø
      RETURN
    2 IF  (C)  3,4,5
    3 W = 5.Ø
      RETURN
    4 W = Ø.Ø
      RETURN
    5 W = C
      RETURN 2
      END
```

Abb. 16.20. Die Anweisung RETURN i

Wird nun in der aufrufenden Programmeinheit das Statement

```
CALL RWY(A,3.4,&1Ø,&2Ø,WAR,&3Ø)
```

ausgeführt, so erfolgt zunächst wie üblich ein ganz normaler Sprung ins Unterprogramm RWY. Die Rückkehr in die aufrufende Programmeinheit ist von den Bedingungen abhängig, die im Unterprogramm geprüft werden. Wenn der Wert der Variablen A negativ ist, so geht die Programmablaufsteuerung zum Statement mit der Marke 1Ø über, wobei WAR auf den Wert 3.4 gesetzt wird. Würde das dem Parameter C entsprechende Argument negativ sein, so würde die Programmablaufsteuerung ganz normal zu dem Statement mit der Marke 4Ø [1] zurückkehren, das unmittelbar hinter der CALL-Anweisung steht (WAR würde dadurch auf den Wert 5.Ø gesetzt werden). Würde schließlich das dem Parameter C entsprechende Argument gleich Ø sein, so würde der Rücksprung vom Unterprogramm zu dem Statement der aufrufenden Programmeinheit erfolgen, das durch die Statementnummer 3Ø gekennzeichnet ist (der Wert von WAR ist hierbei gleich Ø.Ø). Bei diesem speziellen Beispiel entspricht also der im Unterprogramm stehenden Anweisung

```
RETURN i
```

die Anweisungsfolge

1) Wir nehmen hier einmal an, daß das Statement, das auf die CALL-Anweisung der aufrufenden Programmeinheit folgt, durch die Statementnummer 4 gekennzeichnet ist.

```
      CALL RWY(A,CAR,WAR,I)
      GO TO (1Ø,2Ø,3Ø),I
4Ø  ···
```

der aufrufenden Programmeinheit.

Diese Form der RETURN-Anweisung darf nur bei Subroutine-Unterprogrammen verwendet werden; bei Funktionsunterprogrammen besteht diese Möglichkeit nicht.

Ein wohlüberlegter Einsatz der beiden Statements

```
ENTRY          und
RETURN i
```

vereinfacht das Schreiben komplizierter Programme ein wenig. Die Verwendung dieser Statements hat aber andererseits auch zur Folge, daß die Bereinigung von Programmierirrtümern etwas erschwert wird, gerade für den unerfahrenen Programmierer. Man sollte also tunlichst zunächst die Codierung dieser Statements vermeiden, sie also erst dann benutzen, wenn man ein umfangreiches Programm zu schreiben hat und in diesem auf die Logik dieser Statements unbedingt angewiesen ist.

16.6. Die Vereinbarung COMMON

Wie wir bereits gesehen haben, stellen Unterprogramme kleine selbständige Programmeinheiten dar, die entweder zusammen mit dem Hauptprogramm oder getrennt von ihm übersetzt werden können. Ein Objektprogramm, das aus einem in der FORTRAN-Sprache geschriebenen Programm hervorgegangen ist, setzt sich im allgemeinen aus dem Hauptprogramm und aus mehreren Unterprogrammen zusammen; der erfolgreiche Ablauf eines solchen Programmes ist aber nur möglich, wenn der Supervisor des Betriebssystemes sowie die EA-Routinen mitwirken. Wenn der zur Verfügung stehende Hauptspeicherplatz nicht groß genug ist, können einige Programmteile bzw. Programmeinheiten überlagert werden. Oder anders gesehen, einige Programmteile bzw. Programmeinheiten werden erst im Bedarfsfall in den Hauptspeicher geladen. Durch die Vereinbarung COMMON können wir dafür sorgen, daß gewisse Daten ständig verfügbar sind, welche Programmeinheiten auch immer im Hauptspeicher liegen mögen. Mittels dieser Vereinbarung können wir auch auf den Austausch von Argumenten und Parametern zwischen aufrufenden und aufgerufenen Programmeinheiten verzichten.

Der COMMON-Block [1] ist ein spezieller Hauptspeicherbereich, in dem die Daten gespeichert werden können, die von zwei oder mehr Programmen bzw. von einem Programm und seinen Unterprogrammen gemeinsam benutzt werden. Es werden jedoch keine Daten in diesem Bereich gespeichert, wenn das nicht ausdrücklich gefordert wird. Es wird nur jenen Variablen Platz im COMMON-Bereich zugeordnet, die in einer COMMON-Vereinbarung aufgeführt sind.

Das COMMON-Statement besitzt folgende allgemeine Form:

```
COMMON v1,v2,···
```

Jedes v_i bedeutet hierbei eine skalare Variable (d.h. eine nichtindizierte Variable) oder eine Bereichsvariable. Betrachten wir dazu ein Beispiel:

```
COMMON A,B,K,H,M,S
```

1) Ein COMMON-Block ist ein gemeinsam benutzter Hauptspeicherteil, der auch als COMMON-Bereich bezeichnet wird.

Den in dieser Vereinbarung aufgeführten Variablen werden der Reihe nach Hauptspeicherplätze zugeordnet; insgesamt sind es 24, wenn keine expliziten Typvereinbarungen für A,B,K,K,M und S vorliegen. Ein anderes Programm, das mit den gleichen Daten arbeiten soll, könnte dann z.B. das Statement

```
COMMON A,B,LULU,ROT,MARI,ANNE
```

aufweisen. Die Variablen A und B des zweiten Programmes entsprechen den gleichbenannten Variablen des ersten Programmes, so daß man sie ohne Überlegung einander gleichsetzen möchte. Die Namensgleichheit der Variablen in den verschiedenen Programmen bzw. Programmeinheiten ist jedoch keineswegs der entscheidende Punkt, wichtig allein sind die zugeordneten Speicherplätze, d.h. die Reihenfolge derselben. So beziehen sich LULU und K genau so auf die gleichen Speicherplätze wie ROT und H, wie A und A und wie B und B. Damit ist also gesagt, daß z.B. der gleiche Speicherplatz und somit auch der gleiche Wert durch die Variable LULU des zweiten Programmes bzw. durch die Variable K des ersten Programmes angesprochen wird. Es kommt also, das sei noch einmal ausdrücklich betont, nicht auf die Namensgleichheit der einzelnen Variablen in den verschiedenen Programmen bzw. Programmeinheiten an, sondern allein und ausschließlich nur auf ihre Reihenfolge in den einzelnen COMMON-Statements.

Innerhalb einer Programmeinheit darf eine COMMON-Vereinbarung nicht dazu verwendet werden, um eine EQUIVALENCE-Vereinbarung zu ersetzen. Eine Variable darf jedoch in beiden Statements erscheinen, doch hat dann die COMMON-Vereinbarung Vorrang vor der EQUIVALENCE-Vereinbarung. Betrachten wir dazu ein Beispiel:

```
DIMENSION  E(3)
COMMON  A,B,V
EQUIVALENCE  (B,E(1))
```

Im COMMON-Block werden hierdurch den Variablen Speicherplätze in der folgenden Reihenfolge zugeordnet:

```
A
B,E(1)
V,E(2)
  E(3)
```

Es ist nicht erlaubt, mittels einer EQUIVALENCE-Vereinbarung eine Bereichsvariable in den COMMON-Bereich hineinzuzwingen, deren Ausdehnung über die Grenzen des COMMON-Bereiches hinausreichen würde. So ist z.B. die Statementfolge

```
DIMENSION  E(3)
COMMON  A,B,V
EQUIVALENCE  (B,E(3))
```

nicht gestattet, da sie folgende Anordnung der Elemente hervorrufen würde:

```
  E(1)
A,E(2)
B,E(3)
V
```

Da, bedingt durch das COMMON-Statement, der COMMON-Bereich mit A beginnt, würde hier nämlich durch die nachfolgende EQUIVALENCE-Vereinbarung das Bereichselement E(1) jenseits der Grenzen des COMMON-Bereiches zu liegen kommen, also eine ungültige Adresse besitzen. Dem Anfangsprogrammierer sei deshalb dringend ans Herz gelegt, fürs erste keine Variablen sowohl in einer COMMON-Vereinbarung als auch in einer EQUIVALENCE-Vereinbarung aufzuführen.

In ein COMMON-Statement können Variablen der verschiedensten Typen, reelle, ganzzahlige, komplexe und logische, aufgenommen werden. Sie können damit den Variablen einer anderen Programmeinheit entsprechen, sofern diese ebenfalls in ein COMMON-Statement gestellt wurden. Es ist jedoch unerläßlich, daß die Variablen, die einander entsprechen sollen, auch die gleiche Anzahl von Bytes in Anspruch nehmen, d.h. also, daß sich der Programmierer darum kümmern muß, daß die Variablen, die einander entsprechen sollen, auch die gleiche Länge aufweisen. Notfalls muß er eben in die Liste einer COMMON-Vereinbarung Scheinvariable aufnehmen, damit die erforderlichen Begrenzungszuordnungen erreicht werden. So könnte z.B.

```
COMMON A,SCHEI,K,H
```

zusammen mit

```
COMMON A,B,L,H
```

in einer anderen Programmeinheit verwendet werden, um die gewünschte Übereinstimmung der Variablen L und K zu erzwingen. Es ist natürlich nicht notwendig, daß die Variable SCHEI auch im Rest der betreffenden Programmeinheit auftaucht.

Es liegt in der Verantwortung des Programmierers, daß er beim Schreiben des COMMON-Statements selbst auf die richtigen Begrenzungszuordnungen achtet. Bekanntlich werden ja Variablen mit erhöhter (doppelter) Genauigkeit 8 Bytes zugeordnet, deren erstes eine durch 8 teilbare Adresse aufweist, usw. Die einfachste Methode, hier unnötige Fehler zu vermeiden, besteht wohl darin, daß man die Variablen, denen der meiste Hauptspeicherplatz zugeordnet werden muß, zuerst in die Listen der COMMON-Statements hineingestellt. Erfordert z.B. unser Programm die Typvereinbarungen

```
DOUBLE PRECISION  A,H
REAL  HUHU
INTEGER  KARLI
```

so dürfte es am zweckmäßigsten sein, diese Variablen in der folgenden Reihenfolge in ein COMMON-Statement zu stellen:

```
COMMON  A,H,HUHU,KARLI
```

Den Variablen mit der größeren Länge wird somit zuerst Speicherplatz zugeordnet. Damit sind auch die erforderlichen Wortbegrenzungen garantiert.

In COMMON-Statements können auch Informationen über die Dimensionen von Variablen gegeben werden, wodurch die Notwendigkeit ausgeschaltet wird, zusätzliche DIMENSION-Statements für die Variablen des COMMON-Bereiches zu codieren, für die eine Dimensionierung erforderlich ist. Als Beispiel wollen wir annehmen, daß VEKT einen Vektor von 67 Elementen darstellt. Wir würden sinnvollerweise wie folgt codieren:

```
COMMON  HU,VEKT(67),KLM
```

Auf diese Weise ersparen wir uns die Niederschrift einer DIMENSION-Vereinbarung.

Informationen über die Dimensionen von Bereichsvariablen sollten in das Statement aufgenommen werden, in dem die betreffende Variable zum ersten Male erwähnt wird. Das können explizite Typvereinbarungen (REAL, INTEGER usw.), COMMON-Vereinbarungen, aber auch DIMENSION-Vereinbarungen sein. Wenn COMMON-Blöcke sehr umfangreich sind, so erhält man mitunter ein effektiveres Objektprogramm, wenn Bereichsvariable mit vielen Elementen als letzte Variablen in der COMMON-Vereinbarung erwähnt sind.

Das typische Vorgehen beim Schreiben eines umfangreichen Programmes in der FORTRAN-Sprache besteht darin, daß man das Programm in kleinere Teile zerlegt und den ersten Teil als Hauptprogramm entwirft. Jeder Teil ruft dann wieder mehrere

Unterprogramme auf. Stellt man nun die geeigneten Variablen in den COMMON-Bereich, so erübrigt sich die Notwendigkeit, irgendwelche Argumente in den CALL-Anweisungen zu spezifizieren. Abgesehen von dem Vorteil, der sich daraus ergibt, daß man auch Unterprogramme aufrufen kann, die sich nicht ständig im Hauptspeicher befinden, ergibt sich bei Verwendung dieser Methode auch der oft weitaus wichtigere Vorteil, daß man dadurch in die Lage versetzt wird, jeden Teil einzeln zu testen und auf Programmierirrtümer zu untersuchen. Jeder Teil kann außerdem für sich allein Änderungen unterzogen werden, ohne daß man anschließend das gesamte Programm neu übersetzen muß.

In der vollen FORTRAN-Sprache ist es darüberhinaus auch noch möglich, den gesamten COMMON-Bereich in einzelne Blöcke aufzuteilen. Jedem Block kann hierbei ein Name gegeben werden, der den Regeln der FORTRAN-Sprache genügen muß. Ein Teil des gesamten COMMON-Bereiches kann unbenannt bleiben; wir wollen ihn als unbenannten Speicherblock bezeichnen. Die Namen der benannten Speicherblöcke müssen in Schrägstriche eingeschlossen werden; sie stehen jeweils vor der Liste der Variablen, die zu dem betreffenden Block gehören sollen bzw. gehören. Zwei aufeinander folgende Schrägstriche bedeuten, daß der unbenannte Speicherblock wieder in Kraft treten soll, d.h. daß die nachfolgenden Variablen wieder dem unbenannten Block zugeordnet werden sollen. Betrachten wir dazu ein Beispiel:

```
COMMON  A,B/BK1/C,G,K,P/BK2/F,W//RT125
```

Durch diese Vereinbarung werden die Variablen A, B und RT125 in den unbenannten, die Variablen C, G, K und P in den mit BK1 benannten und die Variablen F und W in den mit BK2 benannten Speicherblock gestellt. Benannte Speicherblöcke sind dann von großem Nutzen, wenn der gesamte COMMON-Bereich recht umfangreich ist und sich einige Programme bzw. Programmeinheiten nur auf einen geringen Teil der im gesamten COMMON-Bereich liegenden Daten zu beziehen brauchen.

Wenn mehr als ein COMMON-Statement benutzt wird, so werden alle Statements, selbstverständlich mit Ausnahme des ersten, als Fortsetzung des vorhergehenden Statements angesehen. Das nachfolgende Beispiel möge diese Regel illustrieren:

```
COMMON  /CM/A,B,C//D/CM/E
COMMON  F/CM/G
```

Hier liegen D und F im unbenannten, die Variablen A,B,C,E und G im mit CM benannten COMMON-Block.

16.7. Anfangswerte

Die in diesem Abschnitt zu besprechenden Fakten gelten nur für die volle FORTRAN-Sprache.

Es ist mitunter von Vorteil, wenn gleichzeitig mit dem Laden eines Programmes auch verschiedenen Variablen Anfangswerte zugewiesen werden. In der BASIC FORTRAN-Sprache besteht eine solche Möglichkeit nicht, die volle FORTRAN-Sprache bietet dagegen sogar drei verschiedene Methoden zur Verwirklichung dieser Vorstellungen an.

a) Explizite Typvereinbarungen

Zur Festlegung von Anfangswerten von Variablen können die expliziten Typvereinbarungen REAL, INTEGER, COMPLEX und LOGICAL verwendet werden. Eine diesen Vorhaben dienende Typvereinbarung weist folgenden allgemeinen Aufbau auf:

typ*s v1*s1(k1,···,kn)/r1*x1,···,rm*xm/,v2···

Hierbei ist für typ entweder REAL oder INTEGER oder COMPLEX oder LOGICAL einzusetzen; die Verwendung der Längenangabe *s ist freigestellt. Das gleiche gilt auch für die übrigen im Statement auftretenden Längenangaben *si. Durch die kj (1≤j≤n und n≤7) werden Dimensionsinformationen festgelegt, falls solche erforderlich sind. Durch die rk werden Wiederholungsfaktoren angegeben; diese bewirken, daß rk - mal der nachfolgende xk - Wert den entsprechenden Bereichselementen zugewiesen wird. Betrachten wir zur näheren Erläuterung ein Beispiel:

```
REAL*8  RT,RU*4(3,2)/4.5,3.Ø,2.Ø,1.7,9Ø.2E-Ø3,1.4/,RV/89.Ø/
```

Dieses Statement bewirkt der Reihe nach folgendes: Der Variablen RT werden acht Bytes des Hauptspeichers zugeordnet, für sie ist kein Anfangswert vorgesehen. Der Bereich RU besteht aus sechs Elementen, wobei für jedes Element vier Bytes reserviert werden; den einzelnen Bereichselementen werden folgende Anfangswerte zugewiesen: RU(1,1)=4.5, RU(2,1)=3.0, RU(3,1)=2.0, RU(1,2)=1.7, RU(2,2)=90.2E-03 und RU(3,2)=1.4. Zum Schluß werden schließlich für die Variable RV acht Bytes reserviert, in die beim Laden des betreffenden Programmes der Anfangswert 89.0 gespeichert wird.

b) Das DATA-Statement

Das DATA-Statement ist den expliziten Typvereinbarungen sehr ähnlich; allerdings wird durch dieses Statement kein Typ festgelegt. Das DATA-Statement besitzt folgende Form:

DATA v1,···,vn/r1*x1,···,rm*xm/,vn+1,···

Den Variablen v_1, v_2, ···, v_n werden der Reihe nach die Anfangswerte x_1 (r_1-mal), x_2 (r_2-mal), ···, x_m (r_m-mal) zugewiesen. Die Gesamtzahl der aufgelisten Variablen muß gleich der Gesamtzahl der Anfangswerte sein; die Anfangswerte sind in Schrägstriche einzuschließen, Wiederholungsfaktoren sind erlaubt. Jedes v_i spezifiziert hierbei eine nichtindizierte Variable oder eine Variable mit konstanten Indizes oder den Namen eines Bereiches.

Zur näheren Erläuterung wollen wir uns ein Beispiel ansehen:

```
DATA  RU/4.5,3.Ø,2.Ø,1.7,Ø.Ø9Ø2,1.4/,RV,KW/89.Ø,78/
```

Durch dieses Statement werden den Elementen des Bereiches RU die gleichen Anfangswerte zugewiesen wie bei dem Beispiel, das wir unter a) behandelt haben. Der Variablen RV wird ferner der Anfangswert 89.0 und der Variablen KW der Anfangswert 78 zugewiesen. Die Informationen über die Dimensionen von Bereichen, hier von RU, dürfen nicht in DATA-Statements erscheinen, sie müssen in gesonderten Vereinbarungen aufgeführt werden. Auch die Festlegung von Typen und von Längen für Variable, hier beispielsweise für RV, wenn für diese Variable acht Bytes reserviert werden sollen, muß in gesonderten Statements erfolgen.

c) Das Statement BLOCK DATA

Keiner Variablen, die dem unbenannten Speicherblock des COMMON-Bereiches zugeordnet ist, kann durch irgendeines der bisher besprochenen Statements ein Anfangswert zugewiesen werden; eine andere Möglichkeit besteht bei ihnen nicht.

Spezifikationsunterprogramme besitzen folgenden Aufbau:

```
BLOCK DATA
:
:
END
```

Sie werden also durch die Vereinbarung

```
BLOCK DATA
```

eingeleitet. Mit Hilfe eines einzigen solchen Unterprogrammes können Anfangswerte für Variable eines bekannten Speicherblockes oder mehrerer benannter Speicherblöcke angegeben werden. Ein Spezifikationsunterprogramm darf keine Anweisungen enthalten; es darf vielmehr nur aus Vereinbarungen bestehen, mindestens eine von ihnen muß eine COMMON-Vereinbarung sein. In dieser sind die Variablen aufzuführen, die zu dem betreffenden Speicherblock bzw. zu den betreffenden Speicherblöcken gehören, dabei ist es gleichgültig, ob ihnen Anfangswerte zugewiesen werden sollen oder nicht. Außer COMMON-Vereinbarungen enthält ein BLOCK DATA-Unterprogramm noch die erforderlichen DIMENSION- bzw. expliziten Typvereinbarungen. Es sind natürlich auch DATA-Statements erlaubt, doch werden diese meist nicht benutzt. Wir wollen als Beispiel das Unterprogramm in Abb. 16.21. betrachten:

```
BLOCK DATA
COMMON   /BKNM1/C,G,K,P
REAL*4   C/1.34/,P/1.72/
INTEGER  K/Ø/
END
```

Abb. 16.21. Unterprogramm für Anfangswerte

Man beachte bei dieser Codierung, daß im COMMON-Statement auch die Variable G erscheinen muß, obwohl ihr kein Anfangswert zugewiesen werden soll.

BLOCK DATA - Unterprogramme werden nicht getrennt übersetzt, vielmehr werden sie meist mit dem Hauptprogramm zur Übersetzungszeit zusammengefügt. Sie können natürlich auch mit einem anderen Unterprogramm zusammengefügt werden; in diesem Fall sind die Variablen erst in dem Augenblick definiert, in dem das Unterprogramm aufgerufen wird; sie bleiben solange definiert, bis durch die RETURN-Anweisung die Programmablaufsteuerung wieder der aufrufenden Programmeinheit übergeben wird.

Es sollte noch vermerkt werden, daß die Anfangswerte nur solange erhalten bleiben, bis im Programm den Variablen neue Werte zugewiesen werden. Das kann dadurch geschehen, daß die betreffende Variable in der Datenliste eines Eingabestatements oder auf der linken Seite einer Ergibtanweisung oder als Laufvariable in einer DO-Anweisung oder als Argument in einer CALL-Anweisung erscheint. Der Programmierer sollte deshalb sorgfältig darauf achten, daß keine Variable, deren Anfangswert über das gesamte Programm hinweg erhalten bleiben muß, in einer der o. a. Weisen in irgendwelchen Anweisungen auftritt.

Aufgabengruppe 44

1) Beim Bridgespiel halten zwei Partner 26 Karten. Die restlichen 26 Karten befinden sich in den Händen der beiden Gegenspieler; diese mögen zusammen gerade k Schippen halten. Die Wahrscheinlichkeit W, daß ein bestimmter Gegenspieler $n \leqq k$ Schippen besitzt, läßt sich dann zu

$$W = \frac{C(k, n) \cdot C(26-k, 13-n)}{C(26, 13)}$$

bestimmen. Schreibe ein Programm, daß die Werte von W für $n=1, 2, 3, \cdots, 13$ auf vier Dezimalstellen genau berechnet und ausgibt.

2) Bei der Diskussion im Abschnitt 16.4. nahmen wir an, daß wir nicht unterscheidbare Objekte vor uns hatten. Nehmen wir nunmehr an, daß wir es mit unterscheidbaren Objekten zu tun haben, so ist die Wahrscheinlichkeit dafür, daß es zu m leeren Zellen kommt, gleich

$$P(m;n,r) = \sum_{j=0}^{n-m} (-1)^j\, C(m+j,m)\, C(n,m+j) \left(1-\frac{m+j}{n}\right)^j$$

Schreibe ein Programm, ähnlich dem im Abschnitt 14.4. vorgestellten, zur Berechnung der Werte dieser Funktion.

3) Die Wahrscheinlichkeit dafür, daß man j+n Augen würfelt, wenn man mit n Würfeln wirft, ist gegeben durch

$$P(j;n) = 6^{-n} \sum_{s=0}^{\infty} (-1)^{j+7s}\, C(n,s)\, C(-n, j-6s)$$

Für $j>k>0$ definieren wir $C(k,j)=0$. Für $C(-k,j)$ gilt

$$C(-k,j) = \frac{(-k)\,(-k-1)\cdots(-k-j+1)}{j(j-1)\cdots 2\cdot 1}$$

Schreibe ein Programm, ähnlich dem im Abschnitt 16.4. vorgestellten, zur Berechnung der Werte dieser Funktion.

4) Angenommen, es ist eine Matrix mit den Elementen

$$a_{ij} \quad \text{mit } i = 1,2,\cdots,m \quad \text{und} \quad j = 1,2,\cdots,n$$

gegeben; außerdem sind zwei Vektoren mit den Elementen

$$b_i \quad \text{mit} \quad i = 1,2,\cdots,m \quad \text{und}$$
$$c_j \quad \text{mit} \quad j = 1,2,\cdots,n$$

bekannt.

Das Grundproblem von Linear Programming besteht darin, einen Vektor x_j so zu bestimmen, daß

I. $$\sum_{j=1}^{n} a_{ij}x_j \leqq b_i \quad \text{für } i = 1,2,\cdots,m$$

und

II. $$x_j \geqq 0 \quad \text{für } j = 1,2,\cdots,n$$

ist und daß außerdem

III. $$z = \sum_{j=1}^{n} c_j x_j$$

so groß wie möglich ist.

Schreibe ein Programm in der FORTRAN-Sprache zur Bestimmung der allgemeinen Lösung für $n=2$ und $m \leq 7$. Die Werte für die Matrix- und die Vektorelemente sollen über Karten eingelesen werden. Die Werte der Elemente x_j sowie der Wert von z sind zu stanzen. Alle Eingabewerte sind größer als oder gleich 0.

5) Schreibe ein Programm, daß die Werte für alle Elemente einer Matrix mit drei Zeilen und drei Spalten einliest und anschließend die Determinante dieser Matrix berechnet und ausgibt.

6) Das Programm von Aufgabe 5) ist in ein Unterprogramm umzuformen. Dieses Unterprogramm soll dann in einem Programm aufgerufen werden, daß zur Berechnung der Determinante einer Matrix mit je vier Zeilen und Spalten dienen soll.

   Schreibe auch das Hauptprogramm zur Berechnung der Determinante einer Matrix 4. Ordnung; die Werte der Matrixelemente sind einzulesen und zu drucken. Der Wert der Determinanten ist über die Schreibmaschine auszugeben.

17. Die Einführung in COBOL

17.1. Die Programmierung kommerzieller Probleme

Die kaufmännischen Anwendungen der Computer unterscheiden sich wesentlich von den wissenschaftlichen. Ein typisch wissenschaftliches Problem erfordert umfangreiche Berechnungen auf der Basis weniger Eingabedaten; es werden außerdem verhältnismäßig wenige Ausgabedaten hervorgebracht. Ein typisch kommerzielles Problem hingegen erfordert mehr die Handhabung großer Datenmengen als die Ausführung umfangreicher Berechnungen; meist sind ganze Dateien in die Eingabe bzw. Ausgabe verwikkelt. Bei kaufmännischen Problemen ist das Format der Eingabe- und Ausgabedaten gewöhnlich bis ins einzelne vorgeschrieben und der Programmierer muß die Ausgabe mühselig auf das vorgegebene Format abstimmen. Da die Eingabe meist von bereits existierenden Dateien erfolgt, die entweder auf Karten oder Magnetbändern gespeichert sind, muß er sich überdies mit der Form der Daten auf diesen Datenträgern vertraut machen und sein Programm danach ausrichten. Auf der anderen Seite können bei technisch-wissenschaftlichen Problemen die Daten für beinahe jedes Programm gesondert gelocht werden, so daß der Programmierer selbst das Eingabeformat entwerfen kann, das seinen Vorstellungen am besten entspricht, das er also am bequemsten findet. Da für die Ausgabe im allgemeinen nur die Forderung nach guter Lesbarkeit gestellt wird, braucht er sich außerdem kaum um die Aufbereitung der Daten zu kümmern.

Der Erfolg von FORTRAN in den späten fünfziger Jahren, der sich in einer bemerkenswerten Vereinfachung der Programmierungsarbeiten bei technisch-wissenschaftlichen Problemen niederschlug, ließ viele führende Köpfe nach ähnlichen Methoden suchen, die die Programmierung kaufmännischer Probleme ebenso einfach gestalten konnten. Nach zweijährigen Untersuchungen veröffentlichte 1961 ein vom Verteidigungsministerium der USA gefördertes Komitee, dessen Mitglieder von fast allen Computerherstellern und Benutzern großer Computer delegiert waren, die Spezifikationen einer neuen, für Kaufleute bestimmten problemorientierten Programmiersprache. Diese Sprache wurde COBOL genannt (Abkürzung von Common Business Oriented Language). Die Verwendung der erarbeiteten Unterlagen für Veröffentlichungen jeder Art ist gestattet, es muß freilich der Urhebernachweis vermerkt werden. Wir möchten deshalb an dieser Stelle auf diesen verweisen; er ist im Handbuch „COBOL-1961 Extended" zu finden, das unter der Registriernummer 1962-0668996 von der Regierung der Vereinigten Staaten von Amerika herausgegeben wurde.

COBOL verfolgt drei Ziele:

1) Es soll die Um- bzw. Neuprogrammierung erleichtern, wenn man Programme, die für einen bestimmten Computertyp geschrieben wurden, auf einen anderen Computer übernehmen will oder muß.

2) Es soll die Möglichkeit schaffen, die Durchführung geringer Programmänderungen, mit denen man bei kommerziellen Problemen ständig rechnen muß, einfacher zu gestalten.

3) Es soll die Arbeitszeit verringern helfen, die für das Schreiben und Ausprüfen neuer Programme erforderlich ist.

Das Komitee zog großen Nutzen aus den Erfahrungen, die seine Mitglieder bei der Benutzung anderer problemorientierter Programmiersprachen gewonnen hatten. Deshalb ist COBOL heute die am weitesten verbreitete problemorientierte Sprache auf kommerziellem Gebiet.

Wie die Umgangssprachen entwickeln und ändern sich auch die Programmiersprachen mit der Zeit. Bei COBOL lag der Entwicklungsschwerpunkt in den Jahren 1956 und 1968. Aus den verschiedenen COBOL-Variationen kristallisierte sich 1968 die vom ANS-Institut [1] erarbeitete Version als die gebräuchlichste heraus. Von der IBM wurde sie für den Gebrauch mit den Datenverarbeitungssystemen IBM /360 und IBM /370 sinnvoll und zweckmäßig erweitert. In diesem Buch wollen wir uns deshalb mit dieser Version befassen.

Drei Kompilierer stehen für ANS-COBOL zur Verfügung:

— ANS Subset COBOL für das Betriebssystem DOS
— ANS COBOL für das Betriebssystem DOS
— ANS COBOL für das Betriebssystem OS

Jeder Kompilierer wurde bisher in verschiedenen Versionen herausgebracht, wobei gewöhnlich jede neue Version einige geringfügige Verbesserungen offerierte. Die Kompilierer sind im wesentlichen die gleichen, ausgenommen der Subset-Kompilierer für das Betriebssystem DOS. Bei ihm fehlen verschiedene Einrichtungen, die bei den beiden übrigen Kompilierern vorhanden sind. Wir werden uns vorbehalten, bei der Besprechung der Sprachelemente auf wesentliche Unterschiede zwischen den Kompilierern oder zwischen ihren Versionen ausdrücklich hinzuweisen.

In COBOL basiert die Eingabe/Ausgabe auf Dateien. Jede Datei besteht aus einzelnen Sätzen. Eine Datei wird häufig in verschiedenen Programmen angesprochen, was auf die Bedürfnisse der kaufmännischen Praxis zurückzuführen ist. Die Beschreibung der Dateien muß deshalb sehr präzise sein und darf auch keine Variationen gestatten.

Auf Anregung des Komitees sollte jeder Computerhersteller einen COBOL-Kompilierer herstellen, d.h. ein Programm in der Maschinensprache seines Computers, das als Eingabe in COBOL geschriebene Programme akzeptiert und das diese Programme in Programme übersetzt, denen die Maschinensprache des speziellen Computers zugrunde liegt. Das übersetzte Programm, auch als Objektprogramm bezeichnet, wird dann ausgeführt. Es wurde bei der Entwicklung der Kompilierer auch Vorsorge getroffen, daß sich Quellenmaschine und Objektmaschine unterscheiden können. Dabei versteht man unter Quellenmaschine denjenigen Computer, auf dem die Übersetzung durchgeführt wird; die Objektmaschine ist derjenige Computer, auf dem das übersetzte Programm ablaufen soll. Normalerweise sind Quellenmaschine und Objektmaschine gleich.

Einige Worte noch zur Terminologie: Man spricht von einer Übersetzung, wenn man die Übertragung eines Textes von einer Sprache (hier: COBOL) in eine andere Sprache (hier:Maschinensprache) meint; man redet jedoch von einer Umwandlung, wenn man die Überführung eines Programmes (hier des Quellenprogrammes) in ein anderes Programm (hier in das Objektprogramm) meint.

COBOL unterscheidet sich von den übrigen Programmiersprachen dadurch, daß der Benutzer doch erst einen respektablen Teil der Sprache kennen muß, bevor er in der Lage ist, selbst das einfachste Programm zu schreiben. Dieses Einführungskapitel befaßt sich mit genügend Sprachelementen, so daß man in der Lage sein wird, selbst relativ leichte Programme codieren zu können.

1) ANS ist die Abkürzung von "American National Standards".

17.2. Der Aufbau eines COBOL-Programmes

Die Abb. 17.1. zeigt ein kurzes COBOL-Programm. Jedes COBOL-Programm setzt sich aus vier Teilen zusammen: Erkennungsteil (IDENTIFICATION DIVISION), Maschinenteil (ENVIRONMENT DIVISION), Datenteil (DATA DIVISION) und Prozedurteil (PROCEDURE DIVISION).

Im Erkennungsteil weist der Programmierer zunächst dem Programm einen Namen zu; außerdem können verschiedene weitere Informationen über das Programm, wie der Name des Programmierers und das Erstellungsdatum festgehalten werden. Der Maschinenteil [1] informiert den Kompilierer, über welche Ausrüstungen (Hilfsmittel), wie externe Speichergeräte, EA-Geräte usw. das Objektprogramm verfügen muß. Im Datenteil ist der

```
001010 IDENTIFICATION DIVISION.
001020 PROGRAM-ID. CBL1.
001030 ENVIRONMENT DIVISION.
001040 INPUT-OUTPUT SECTION.
001050 FILE-CONTROL.
001060     SELECT CD ASSIGN TO UR-2540R-S-CD.
001070     SELECT TT ASSIGN TO UT-2400-S-TAPE.
001080 DATA DIVISION.
001090 FILE SECTION.
001100 FD  CD       DATA RECORD IS C,
001110              LABEL RECORDS ARE OMITTED, RECORDING MODE IS F.
001120 01  C.
001130     02  C1   PICTURE 9(4).
001140     02  C2   PICTURE 9.
001150     02  C3   PICTURE X(70).
001160     02  C4   PICTURE X(5).
001170 FD  TT       DATA RECORD IS T,
001180              LABEL RECORDS ARE OMITTED, RECORDING MODE IS F.
001190 01  T       PICTURE X(160).
001200 WORKING-STORAGE SECTION.
001210 01  D.
001220     02  D2   PICTURE S9      COMPUTATIONAL.
001230     02  D1   PICTURE 9(4).
001240     02  D3   PICTURE X(70).
001250     02  D4   PICTURE X(70).
001260     02  D5   PICTURE X(6)    VALUE '010167'.
001270     02  D6   PICTURE S(7)V99 VALUE ZERO      COMPUTATIONAL-3.
001280     02  D7   PICTURE XXX     VALUE SPACE.
001290 PROCEDURE DIVISION.
001300 P1. OPEN INPUT CD, OUTPUT TT.
001310 P2. READ CD, AT END GO TO P3.
001320     MOVE C1 TO D1.
001330     MOVE C2 TO D2.
001340     MOVE C3 TO D3.
001350     READ CD, AT END GO TO P3.
001360     MOVE C3 TO D4.
001370     WRITE T FROM D.
001380     GO TO P2.
001390 P3. CLOSE CD, TT.
001400     STOP RUN.
```

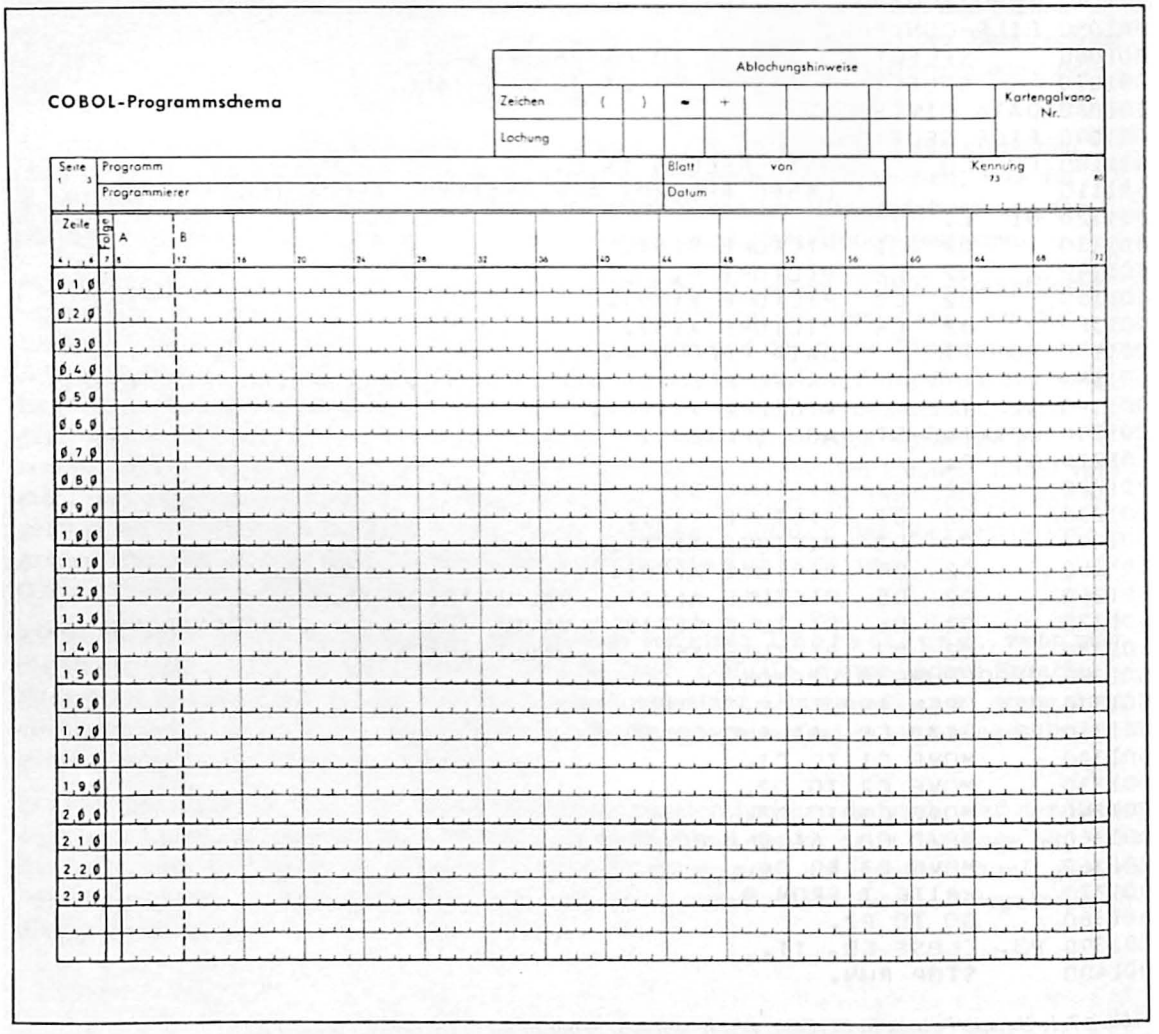
Abb. 17.1. Beispiel eines in COBOL geschriebenen Programmes (Erstellung einer Datei)

1) Der Maschinenteil wird auch als Ausrüstungsteil bezeichnet.

Aufbau und die Zusammensetzung jeder Datei exakt zu beschreiben; das gleiche gilt auch für jedes benutzte Arbeitsgebiet. Der Prozedurteil enthält schließlich die Anweisungen für die Eingabe/Ausgabe und für die Verarbeitung. Alle vier Teile müssen stets in der genannten Reihenfolge erscheinen (siehe Abb. 17.1.).

Beim Wechsel von einem Computer zu einem anderen braucht im allgemeinen nur der Maschinenteil abgeändert zu werden. Wenn eine Datei in mehr als einem Programm angesprochen wird, so kann weiterhin ihre im Datenteil vorgenommene Beschreibung von einem Programm in ein anderes übertragen werden; auf diese Weise kann die Zeit reduziert werden, die man sonst für das Schreiben neuer Programme benötigen würde.

Das COBOL-Programm wird gewöhnlich auf spezielle Programmschemen geschrieben; ein solches wird in Abb. 17.2. gezeigt. Jede Zeile des Schemas wird auf eine 80spaltige Standardlochkarte übertragen. Meist stehen außerdem Lochkarten mit einem speziellen Aufdruck zur Verfügung. Die abgelochten Karten lassen sich dadurch leichter überprüfen (siehe hierzu Abb. 17.3.).



Anmerkung: Beim Niederschreiben von Programmen wird gewöhnlich die Null durchgestrichen, um Verwechslungen mit dem Buchstaben O auszuschließen.

Abb. 17.2. COBOL-Programmschema

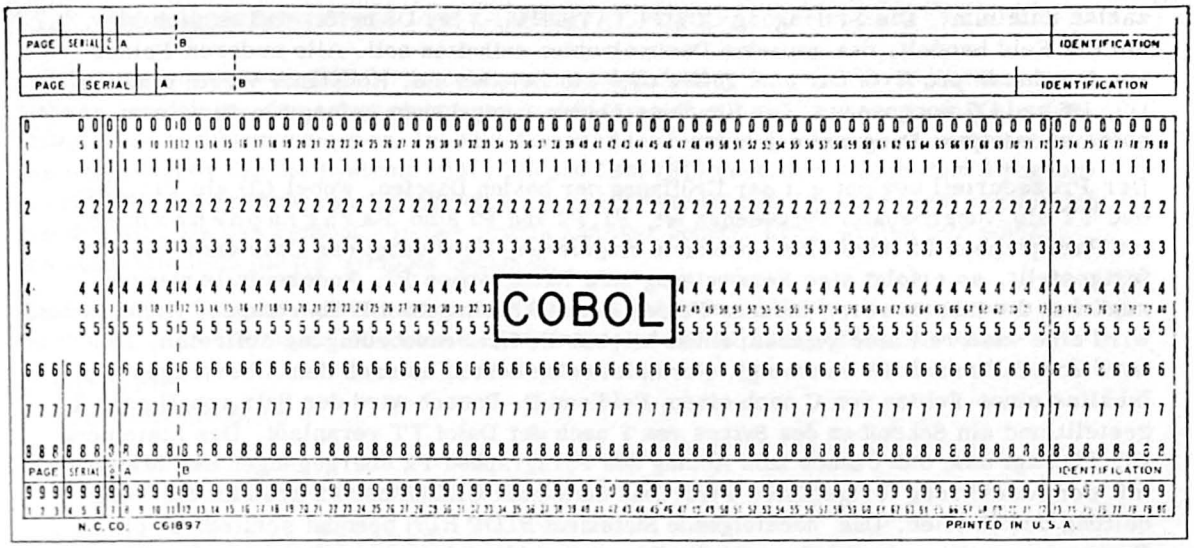


Abb. 17.3. Muster einer speziell zum Ablochen von COBOL-Programmen gedachten Lochkarte

Verfolgen wir nun die Einteilung des COBOL-Programmschemas im Detail.

Auf den Spalten 1 bis 6 wird die Seiten- und die Zeilenzahl festgehalten. Daraus ergeben sich in erster Linie nur Vorteile für den Programmierer. Notwendig ist jedenfalls die Ausfüllung und Ablochung dieser Spalten nicht. Man kann allerdings vom Kompilierer auch verlangen, daß er die in diesen Spalten stehenden Zahlen auf aufsteigende Reihenfolge hin prüft. Gewöhnlich vergibt man die Zeilennummern zunächst nicht lückenlos fortlaufend, damit man später Einfügungen vornehmen kann, d.h. z.B. Ø1Ø, Ø2Ø, Ø3Ø usw.

Die Spalte 7 bleibt normalerweise leer. Ein Bindestrich (Minuszeichen) in dieser Spalte zeigt an, daß gewisse Konstanten von der vorhergehenden Zeile fortgesetzt sind.

Die COBOL-Statements werden so niedergeschrieben, daß sie entweder zwischen den Spalten 8 und 11 einschließlich (Randbegrenzung A) oder irgendwo zwischen den Spalten 12 und 72 einschließlich (Randbegrenzung B) beginnen. Welche Randbegrenzung benutzt werden muß, hängt gänzlich von der Eintragung ab, d.h. vom Statementtyp. Die Statements, die bei der Randbegrenzung A anfangen, können selbstverständlich über diese hinaus bis zur Spalte 72 einschließlich fortgesetzt werden.

Die Spalten 73 bis 80 werden von den Kompilierern nicht beachtet. Sie enthalten vielfach den Namen des Programmes.

Die weitere Besprechung von COBOL wollen wir erst einmal unterbrechen, um das Beispiel von Abb. 17.1. zu untersuchen. Der Erkennungsteil teilt uns mit, daß das Programm den Namen CBL1 besitzt. Im Maschinenteil sind zwei Dateien definiert: CD ist einem Kartenleser des Typs IBM 2540 und TT einem Magnetbandgerät zugeordnet. Im Datenteil ist festgelegt, daß die Datei CD aus achtzigstelligen Sätzen C besteht, die in ein vierstelliges Feld C1, ein einstelliges Feld C2, ein siebzigstelliges Feld C3 und ein fünfstelliges Feld C4 zerfallen. Die Datei TT enthält 160-stellige Sätze T, die nicht weiter unterteilt sind. Das Arbeitsgebiet besteht nur aus einem einzigen Satz D, der in die sieben Felder D2, D1, D3, D4, D5, D6 und D7 unterteilt ist. Die Eintragung COMPUTATIONAL bei D2 bedeutet, daß dieses Feld ein Halbwort darstellt, das Dual-

zahlen aufnimmt. Die Eintragung COMPUTATIONAL-3 bei D6 heißt, daß es sich hier um ein Feld handelt, das gepackte Dezimalzahlen enthalten soll. Alle anderen Felder von D nehmen pro Byte nur eine Ziffer oder ein Zeichen auf. Konstante Werte sind für D5, D6 und D7 vorgesehen. Die für diese Felder angegebenen Anfangswerte werden zusammen mit dem Programm geladen.

Der Prozedurteil beginnt mit der Eröffnung der beiden Dateien, wobei CD als Eingabe- und TT als Ausgabedatei vorgesehen ist. P1, P2 und P3 sind Paragraphennamen. Im Paragraphen P2 wird eine Lochkarte eingelesen. Wird dabei das Dateiende von CD festgestellt, so erfolgt eine Verzweigung zum Paragraphen P3. Anderenfalls werden zunächst die Inhalte verschiedener Felder in das Arbeitsgebiet D übertragen. Anschließend wird eine weitere Karte gelesen; sollte hierbei die Dateiendebedingung auftreten, so wird ebenfalls nach P3 verzweigt. Normalerweise kommt es nach dem Übertragen des Inhaltes eines Feldes von C nach einem Feld von D. Danach wird der Satz von D nach T gestellt und ein Schreiben des Satzes von T nach der Datei TT veranlaßt. Das Statement GO TO sagt uns, daß danach zum Anfang des Paragraphen P2 übergegangen und hierauf mit der Ausführung fortgefahren wird. Das Statement CLOSE bei P3 beendet die Verarbeitung der Dateien. Das nachfolgende Statement STOP RUN beendet schließlich die Programmausführung und übergibt die Programmablaufsteuerung zurück an den Supervisor.

Das Programm liest also in einem Zyklus (einer Schleife) zwei Lochkarten und stellt aus ihrem Inhalt einen Bandsatz zusammen. Dem Inhalt der Karte kommt dabei folgende Bedeutung zu:

a) *Karte 1*

Spalten 1 bis 4:	Kundennummer
Spalte 5 bis :	Preisklasse
Spalten 6 bis 40:	Kundenname
Spalten 41 bis 75:	Adresse (1. Zeile)
Spalten 76 bis 80:	leer

b) *Karte 2*

Spalten 1 bis 5:	leer
Spalten 6 bis 40:	Adresse (2. Zeile)
Spalten 41 bis 75:	Adresse (3. Zeile)
Spalten 76 bis 80:	leer

Die aufgezeichneten Bandsätze sind dann wie folgt zusammengesetzt:

Stellen 1 und 2:	Preisklasse
Stellen 3 bis 6:	Kundennummer
Stellen 7 bis 41:	Kundenname
Stellen 42 bis 76:	Adresse (1. Zeile)
Stellen 77 bis 111:	Adresse (2. Zeile)
Stellen 112 bis 146:	Adresse (3. Zeile)
Stellen 147 bis 152:	Datum in der Form ttmmjj (Tag-Monat-Jahr)
Stellen 153 bis 157:	Kontenstand, auf 0.00 DM gesetzt
Stellen 158 bis 160:	Leerzeichen

17.3. Der Zeichenvorrat von COBOL

Der Zeichenvorrat von COBOL besteht aus nur 51 Zeichen. Er schließt 26 Großbuchstaben, 10 Ziffern, das Leerzeichen und 14 Sonderzeichen ein. Zu diesen zählt als Spezialfall das Anführungszeichen. Im ANS-Standard ist hierfür das Symbol " vorgesehen. Die Kompilierer für die Systeme IBM /360 und IBM /370 gestatten jedoch dem Programmierer die Verwendung von ' oder von " als Anführungszeichen, jedoch nicht von beiden gleichzeitig. Da das Symbol ' in der Praxis häufiger benutzt wird, wollen wir selbst uns ausschließlich dieses Symbols bedienen.

Wie bei jeder Sprache gibt es auch in COBOL Wörter, deren definierte Bedeutung von ihrer normalen abweicht. Folgende Definitionen sind für COBOL getätigt worden:

a) Ziffern
0 bis einschließlich 9

b) Buchstaben
Großbuchstaben A bis einschließlich Z

c) Alphabetisches Zeichen
Großbuchstaben A bis einschließlich Z sowie das Leerzeichen (Zwischenraumzeichen)

d) Alphanumerisches Zeichen
Jedes Zeichen der 256 möglichen Zeichen des EBCDI-Codes

e) Numerisches Zeichen
0 bis einschließlich 9 (wie Ziffern)

f) Sonderzeichen
Zu den 14 Sonderzeichen gehören

+	Pluszeichen
-	Minuszeichen oder Bindestrich
*	Sternzeichen
/	Schrägstrich oder Divisionszeichen
=	Gleichheitszeichen
,	Komma
;	Semikolon
$	Währungszeichen (hier: Dollarzeichen)
.	Punkt
' oder "	Anführungszeichen
(	Geöffnete Klammer
)	Geschlossene Klammer
>	Größer-als-Zeichen
<	Kleiner-als-Zeichen

17.4. Wörter und Literale in COBOL

Unter einem Wort versteht man eine Folge von nicht mehr als dreißig Buchstaben, Ziffern und Minuszeichen, die mit einem Buchstaben oder einer Ziffer beginnt und endet. Beispiele für COBOL-Wörter sind:

```
X77B
RECHNUNGS_BETRAG
8
7H--7
IFA-BK-35Ø
```

Nachfolgend sind einige Zeichenfolgen aufgeführt, die nicht als Wörter gelten können:

```
-7      (beginnt nicht mit einem Buchstaben oder einer Ziffer)
H2$M    (enthält ein illegales Zeichen)
7yU     (enthält ein illegales Zeichen)
```

Einem Wort geht stets ein Leerzeichen oder eine geöffnete Klammer voraus; ihm folgt entweder ein Leerzeichen oder eine geschlossene Klammer oder ein Punkt oder ein Komma oder ein Semikolon.

Als reserviertes Wort gilt jedes Wort, das in eine Spezialliste von Wörtern aufgenommen ist, deren Benutzung in COBOL einer Beschränkung unterworfen ist. Eine Liste von reservierten Wörtern ist im Anhang G zu finden. Diese Wörter dürfen nur für Zwecke verwendet werden, die ihrer Bedeutung entsprechen. In die genannte Liste sind natürlich auch solche Wörter wie ZERO, SPACE usw. aufgenommen.

Unter einem Datennamen versteht man ein Wort, das mindestens einen Buchstaben in seiner Zeichenfolge aufweist. Somit gelten 8H9 und IFA-BK-35Ø als Datennamen, 796 jedoch nicht als ein solcher.

Als Literal ist eine Zeichenfolge anzusehen, die in sich selbst einen Wert einschließt. Es gibt zwei Gruppen von Literalen:

a) numerische Literale
b) nichtnumerische Literale

Bei den numerischen Literalen muß zwischen den Festpunktliteralen und den Gleitpunktliteralen unterschieden werden.

Numerische Festpunktliterale bestehen aus 1 bis 18 Ziffern, denen ein Pluszeichen oder ein Minuszeichen vorausgehen kann, aber nicht muß. Fehlt ein Vorzeichen, so wird das Pluszeichen unterstellt. Ein Festpunktliteral kann zur Kenntlichmachung der Stellung des Dezimalpunktes zusätzlich einen Punkt enthalten. Fehlt ein solcher, so wird ein solcher unmittelbar rechts neben der Ziffer niedrigster Ordnung angenommen. Nachfolgend einige Beispiele für Festpunktliterale:

```
-3277
+38.
24.81973
```

Die beiden nachstehenden Zeichenfolgen gelten nicht als Festpunktliterale:

```
-3,277  (enthält ein illegales Zeichen)
3277-   (das Vorzeichen geht nicht der ersten Ziffer voraus)
```

Ein numerisches Gleitpunktliteral besteht aus der Mantisse, dem Buchstaben E und dem Exponenten. Die Mantisse setzt sich aus 1 bis 16 Ziffern zusammen; ein Dezimalpunkt kann hinzutreten. Ein Vorzeichen (plus oder minus) kann zusätzlich vorausgehen. Der Exponent besteht aus einer Ziffer oder aus zwei Ziffern, ihm kann ebenfalls ein Vorzeichen (plus oder minus) vorangestellt werden. Nachfolgend einige Beispiele für Gleitpunktliterale:

```
-24,8E+2
+19.73E3
33685E+ØØ
-1.9E-25
```

Ein nichtnumerisches Literal besitzt als Wert eine Zeichenkette. Es besteht aus 1 bis 120 alphamerischen Zeichen mit Ausnahme des Anführungszeichens und wird von zwei

Anführungszeichen umgeben. Die einschließenden Anführungszeichen gehören selbst nicht zum Literal. Nachfolgend einige Beispiele für nichtnumerische Literale:

Literal	Schreibweise
796	'796'
MARIANNE	'MARIANNE'
Literal 3	'Literal 3'

Die beiden nachfolgenden Beispiele gelten wegen der in ihnen vorkommenden Anführungszeichen nicht als nichtnumerische Literale:

```
'GEHT'S EUCH NOCH GUT?'
'AIN''T'
```

Unter einer figurativen Konstanten [1] versteht man ein Literal, das durch einen speziellen Datennamen ausgezeichnet ist. Zu den figurativen Konstanten zählen:

a) ZERO,ZEROS oder ZEROES
Diese Konstante repräsentiert eine Null oder mehrere Nullen.

b) SPACE oder SPACES
Durch diese Konstante wird ein Leerzeichen oder mehrere Leerzeichen dargestellt.

c) QUOTE oder QUOTES
Diese Konstante repräsentiert ein oder mehrere Anführungszeichen.

d) HIGH-VALUE oder HIGH-VALUES
Diese Konstante repräsentiert ein oder mehrere Zeichen, deren Bytes auf allen Bitstellen Einsen aufweisen, d.h. sedezimal FF darstellen.

e) LOW-VALUE oder LOW-VALUES
Diese Konstante repräsentiert ein oder mehrere Zeichen, deren Bytes auf allen Bitstellen Nullen aufweisen, d.h. sedezimal 00 darstellen.

f) ALL'x'
Diese Konstante repräsentiert das ein- oder mehrmalige Auftreten des Zeichens x, wobei x irgendein alphamerisches Zeichen mit Ausnahme des Anführungszeichens darstellen kann.

Zum Schluß wollen wir noch auf folgende Tatsache hinweisen: Eine besondere Festlegung erlaubt den Austausch des Punktes und des Kommas in bestimmten Teilen von COBOL-Programmen, wozu auch die numerischen Literale gehören.

17.5. Interpunktionsregeln

COBOL erfordert die Beachtung einiger Interpunktionsregeln. Diese Regeln gelten nicht für die Zeichen in nichtnumerischen Literalen und auch nicht für Dezimalpunkte.

Interpunktionsregeln

1) Einem Punkt, einem Komma oder einem Semikolon darf kein Leerzeichen vorausgehen; es muß ihnen jedoch ein Leerzeichen folgen.

2) Wenigstens ein Leerzeichen muß aufeinanderfolgende Wörter, Literale und Ausdrücke voneinander trennen.

3) Zwei oder mehr Leerzeichen werden wie ein einziges Leerzeichen betrachtet.

---

1) Auch als figürliche oder vorgesehene Konstanten bezeichnet.

4) Einer geöffneten Klammer darf nicht unmittelbar ein Leerzeichen folgen.

5) Einer geschlossenen Klammer darf kein Leerzeichen vorausgehen.

6) Einem unären Plus- oder Minuszeichen, d.h. also einem positiven oder negativen Vorzeichen, muß ein Leerzeichen oder eine geöffnete Klammer vorausgehen.

7) Einem Gleichheitszeichen und binären arithmeitschen Operatoren, d.h. + usw., müssen Leerzeichen sowohl vorausgehen als auch nachfolgen.

Verstöße gegen diese Regeln werden von den Kompilierern durch die Ausgabe von Meldungen während der Übersetzung geahndet. In einigen Fällen werden die Kompilierer eine Korrektur der Fehler versuchen. Es ist jedoch immer besser, solche Regelverstöße zu vermeiden als mit ihnen zu beginnen.

17.6. Hinweise zur Notation

Bei der Erklärung der COBOL-Statements wollen wir uns an einige Konventionen bezüglich der Schreibweise halten, die universell gebraucht werden.

1) Schlüsselwörter sind Wörter mit einer festen Bedeutung, die so niedergeschrieben werden müssen, wie sie in den Erklärungen aufgeführt sind. Sie bestimmen die Art des Statements oder einen speziellen Teil desselben. Sie werden nur in Großbuchstaben geschrieben und außerdem unterstrichen.

2) Optionale Wörter sind Wörter mit einer festen Bedeutung, die niedergeschrieben werden können; sie können aber auch weggelassen werden. Ihre Verwendung dient nur dem Zweck, die Lesbarkeit des Quellenprogrammes zu verbessern. Solche Wörter sollen in den Erklärungen ebenfalls in Großbuchstaben geschrieben werden, allerdings ohne Unterstreichung.

3) Informationen, die vom Programmierer nach seinem Ermessen hinzuzustellen sind, erscheinen in den Erklärungen in Kleinbuchstaben und unterstrichen.

4) Sonderzeichen müssen so niedergeschrieben werden, wie sie in den Erklärungen gebraucht sind. Sie werden nämlich als Interpunktionszeichen benötigt. Das gilt auch für solche, die nicht unterstrichen sind.

5) Eckige Klammern [ ] zeigen an, daß das in ihnen enthaltene Element ein hinsichtlich der Benutzung freigestelltes Element ist. Umschließen eckige Klammern mehrere übereinanderstehende Elemente, so kann unter ihnen ein beliebiges ausgewählt werden. Es kann aber auch auf alle Elemente verzichtet werden.

6) Geschweifte Klammern { } zeigen an, daß das in ihnen enthaltene Element unbedingt benutzt werden muß. Umschließen geschweifte Klammern mehrere übereinanderstehende Elemente, so muß eines und nur eines dieser Elemente verwendet werden.

7) Drei aufeinanderfolgende Punkte in Zeilenmitte ··· zeigen an, daß die ihnen unmittelbar vorangehende Information in beliebiger Anzahl wiederholt werden kann. Geschweifte Klammern können zur Kenntlichmachung der Information herangezogen werden, die wiederholt werden darf.

Die soeben genannten Schreibhinweise für die Erklärungen sollen durch ein Beispiel untermauert werden.

$$\underline{\text{DISPLAY}} \left\{ \begin{matrix} \underline{\text{literal-1}} \\ \underline{\text{bezeichnung-1}} \end{matrix} \right\} \left[\begin{matrix} \underline{\text{literal-2}} \\ \underline{\text{bezeichnung-2}} \end{matrix} \right] \cdots \; [\text{UPON } \underline{\text{mn}}]$$

Das Wort DISPLAY muß als Schlüsselwort niedergeschrieben werden; nur durch sein Auftreten kann die Art des Statements festgestellt werden. Auf DISPLAY muß entweder ein vom Programmierer genanntes Literal literal-1 oder eine von ihm genannte Bezeichnung bezeichnung-1 folgen. (Wir werden später zu gegebener Zeit den Begriff „Bezeichnung" erklären.) Zusätzliche Literale und/oder Bezeichnungen können sich anschließen. Die Benutzung der Kausel UPON ist freigestellt. Wird sie jedoch verwendet, so muß UPON niedergeschrieben werden. Man beachte hierzu, daß das Wort UPON die Literale bzw. die Bezeichnungen von dem mnemonischen Namen mn trennt.

17.7. Der Erkennungsteil

Der gesamte Erkennungsteil eines COBOL-Programmes kann das in Abb. 17.4. dargestellte Aussehen annehmen.

```
{IDENTIFICATION}
{ID            } DIVISION.
 PROGRAM-ID. programmname.
[AUTHOR. [kommentar.]]
[INSTALLATION. [kommentar.]]
[DATE-WRITTEN. [kommentar.]]
[DATE-COMPLIED. [kommentar.]]
[SECURITY. [kommentar.]]
[REMARKS. [kommentar.]]
```

Abb. 17.4. Allgemeiner Aufbau des Erkennungsteiles

Jeder Paragraphenname beginnt innerhalb der Randbegrenzung A. Jede Eintragung, die als Kommentar angesehen wird, besteht aus einem oder mehreren Sätzen und muß vollständig innerhalb der Stellen niedergeschrieben werden, die bei der Randbegrenzung B anfangen. Die einzigen Regeln, die bei der Niederschrift von Kommentaren zu beachten sind, sind die, daß die Sätze den Zeichensetzungsregeln von COBOL genügen müssen. Die einzelnen Paragraphen können in einer beliebigen Reihenfolge codiert werden, doch die hier dargestellte wird offensichtlich bevorzugt. An Stelle von IDENTIFICATION kann auch die Abkürzung ID benutzt werden.

Für programmname kann irgendein beliebiges gültiges COBOL-Wort geschrieben werden. Es ist jedoch unbedingt anzuraten, daß er auf nicht mehr als acht Zeichen beschränkt wird und daß das erste Zeichen ein Buchstabe ist.

Aus der unter programmname vom Programmierer festgelegten Information erzeugt der Kompilierer den Namen, unter dem das Programm dem Betriebssystem bekannt ist. Überschreitet der Name acht Zeichen, so werden nur die ersten acht benutzt. Beginnt der Name nicht mit einem Buchstaben, so wird das erste Zeichen in einen Buchstaben übergeführt, wobei folgende Regeln gelten:

— 0 wird in J übergeführt
— 1 bis 9 werden entsprechend in A bis I übergeführt

Minuszeichen (Bindestriche) innerhalb des vom Programmierer eingeführten Namens werden in Nullen umgewandelt.

Abgesehen vom Paragraphen DATE-COMPILED werden die unter kommentar aufgeführten Bemerkungen einfach in den Programmauflistungen reproduziert, bilden also letzten Endes nur eine Dokumentationshilfe. Die unter kommentar beim Paragraphen DATE-COMPILED aufgeführten Bemerkungen werden bei der Übersetzung entfernt und durch das Datum des Tages, an dem die Übersetzung vorgenommen wird, ersetzt.

Für den Erkennungsteil geben wir abschließend in Abb. 17.5. ein typisches Beispiel.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.   GANDIA.
REMARKS.   FORTSETZUNG VON BEMERKUNGEN
    ERST INNERHALB DER RANDBEGRENZUNG
    B MOEGLICH.
```

Abb. 17.5. Typisches Beispiel für den Erkennungsteil eines COBOL-Programmes

17.8. Der Maschinenteil

Die grundlegenden Eintragungen im Maschinenteil lauten:

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   {SELECT satz.} ...
```

Der Teilname ENVIRONMENT, der Kapitelname INPUT-OUTPUT und der Paragraphenname FILE-CONTROL beginnen innerhalb der Randbegrenzung A. Die SELECT-Eintragungen müssen vollständig auf den der Stellen notiert werden, die erst bei der Randbegrenzung B anfangen. Je eine solche Eintragung muß für jede Datei, die im Programm angesprochen wird, niedergeschrieben werden. Dabei muß ein Auswahlsatz (SELECT-Satz) die folgende Form aufweisen:

```
SELECT dateiname ASSIGN TO systemname.
```

Es ist also zumindest für jede Datei die sogenannte Zuweisungsklausel (ASSIGN-Klausel) niederzuschreiben. Unter dateiname ist hierbei der Name zu verstehen, unter dem eine Datenmenge im Quellenprogramm angesprochen wird. [1)]

Im Betriebssystem OS weist systemname die folgende Form auf:

```
klasse [-gerät] -organisation [-name]
```

Im Betriebssystem DOS sieht systemname hingegen wie folgt aus:

```
SYSnnn-klasse-gerät-organisation [-name]
```

Durch SYSnnn mit nnn gleich ØØØ bis Ø21 wird im DOS die logische Einheit bezeichnet, zu der die Datei gehört. Man beachte dabei, daß nur die logischen Einheiten spezifiziert werden können, die vom Programmierer vergeben sind.

1) Eine in einem Quellenprogramm angesprochene Datenmenge wird bekanntlich als Datei bezeichnet.

Durch klasse wird die Kategorie des Gerätetyps wie folgt festgelegt:

DA	Externe Speichergeräte mit direkter Zugriffsmöglichkeit
UT	Hilfsgeräte
UR	Satzeinheitengeräte [1]

Unter gerät ist der spezielle Gerätetyp zu nennen, wobei folgende Möglichkeiten bestehen:

klasse	gerät
DA	2311, 2314, 23Ø5 (nur im OS), 333Ø, 334Ø
UT	24ØØ (für alle Magnetbandgeräte) sowie alle Geräte, die unter DA aufgeführt sind
UR	Zeilendrucker: 14Ø3, 1443, 1445, 3211 Lochkartenleser: 1442R, 25Ø1, 252ØR, 254ØR, 254ØP (Stanzen in gleiche Karte), 35Ø5 Lochkartenstanzer: 252ØP, 254ØP, 3525P

Unter organisation steht S bei konsekutiv organisierten Dateien.

Unter name ist der ddname im OS bzw. der Name des TLBL- bzw. DLBL-Statements im DOS aufzuführen. Bei Dateien, die im DOS ohne Kennsätze (Etiketten) vorliegen, ist dieser Name nicht zu verwenden. Wird im DOS sonst der Name weggelassen, so wird an seiner Stelle der Name SYSnnn benutzt. Der Name muß im übrigen den Erfordernissen des OS bzw. DOS genügen, d.h. mit einem Buchstaben beginnen, nur Buchstaben und Ziffern aufweisen usw.

Wird im Betriebssystem OS unter gerät keine Angabe gemacht, so kann der Gerätetyp im DD-Statement der Jobsteuersprache festgelegt werden, wodurch eine Geräteunabhängigkeit im Rahmen der durch die Klasse abgesteckten Grenzen erzielt werden kann.

In Abb. 17.6. ist für den Maschinenteil ein typisches Beispiel aufgeführt.

Betriebssystem DOS	*Betriebssystem OS*
ENVIRONMENT DIVISION. INPUT-OUTPUT SECTION. FILE-CONTROL. SELECT CD ASSIGN TO SYSØØ4-UR-35Ø5-S. SELECT PD ASSING TO SYSØØ6-UR-254ØP-S-PCH.	ENVIRONMENT DIVISION. INPUT-OUTPUT SECTION. FILE-CONTROL. SELECT CD ASSIGN TO UR-35Ø5-S-SYSIN. SELECT PD ASSIGN TO UR-254ØP-S-SYSPUNCH.

Abb. 17.6. Typisches Beispiel für den Maschinenteil eines COBOL-Programmes in den Betriebssystemen DOS und OS

Die beiden Codierungen von Abb. 17.6. verweisen auf zwei Dateien:

a) CD ist der Dateiname einer Lochkarteneingabedatei
b) PD ist der Dateiname einer Lochkartenausgabedatei (Stanzdatei)

1) Als Satzeinheitengerät gelten diejenigen Geräte, die zu Sätzen zusammengefaßte Informationen mechanisch auf Datenträger aufzeichnen oder denen derart aufgezeichnete Informationen zugeführt werden.

17.9. Die Dateibeschreibung des Datenteils

Die grundlegenden Eintragungen des Datenteils lauten wie folgt:

```
DATA DIVISION.
FILE SECTION.
    eintragungen.
[WORKING-STORAGE SECTION.]
    [eintragungen]
```

Eintragungen im Dateienkapitel müssen für jede im Programm angesprochene Datei vorhanden sein. Eintragungen im Arbeitsspeicherkapitel müssen für jedes Datenelement vorliegen, das nicht in die EA verwickelt ist.

Die Wörter DATA, FILE und WORKING-STORAGE müssen innerhalb der Randbegrenzung A anfangen.

Im Dateienkapitel ist die Struktur jeder im Programm benutzten Datei zu definieren. Im Arbeitsspeicherkapitel hingegen wird das Format derjenigen Sätze und Datenelemente festgelegt, die nicht Bestandteil von Datein sind. Sind keine solchen Sätze und Elemente vorhanden, so kann das Arbeitsspeicherkapitel entfallen.

Der erste, für jede Datei niederzuschreibende Satz besitzt folgendes Aussehen:

```
FD  dateiname
    [BLOCK CONTAINS [g1 TO] g2 {CHARACTERS}]
                               {RECORDS   }
    [RECORD CONTAINS [g3 TO] g4 CHARACTERS]
    [RECORDING MODE IS modus]
     LABEL {RECORD IS  } {OMITTED }
           {RECORDS ARE} {STANDARD}
    [DATA {RECORD IS  }
          {RECORDS ARE} dn1 [dn2] ...] .
```

FD muß innerhalb der Randbegrenzung A beginnen. Der Rest des Satzes muß gänzlich innerhalb der durch die Randbegrenzung B vorgegebenen Stellen der Zeilen liegen. Der unter dateiname aufgeführte Name muß der gleiche sein, der im Maschinenteil zugeordnet wurde. Die verschiedenen Satzteile können in einer beliebigen Reihenfolge angeordnet werden. Von allen Satzteilen ist nur der durch LABEL eingeleitete notwendiger Bestandteil des Satzes.

Fehlt die Klausel BLOCK, so werden ungeblockte Sätze vorausgesetzt. Durch BLOCK wird im übrigen die physikalische Blocklänge festgehalten, einschließlich evtl. vorhandener Steuerbytes. Wird g1 weggelassen, so wird g1 gleich g2 angenommen. Wird RECORDS nicht niedergeschrieben, so wird CHARACTERS unterstellt.

Die Klausel RECORD dient zur Spezifizierung der minimalen (g3) und der maximalen (g4) Länge der Sätze innerhalb der Datei. Wird g3 weggelassen, so wird g3 gleich g4 angenommen. Gleichgültig, ob diese Klausel niedergeschrieben oder weggelassen wird, der Kompilierer entnimmt stets die Satzlängen aus den nachfolgenden Satzbeschreibungen. Für g1, g2, g3 und g4 sind grundsätzlich nur ganze Zahlen einzusetzen.

Die Klausel RECORDING dient zum Festlegen der Satzart. Für modus können die folgenden Kennbuchstaben niedergeschrieben werden:

F	Sätze fester Länge
V	Sätze variabler Länge
U	Sätze undefinierter Länge
S	Sätze im S-Format (Spannsätze)

Wird diese Klausel weggelassen, so werden Sätze variabler Länge (Satzart V) unterstellt.

Die Klausel LABEL ist erforderlich. Das Wort OMITTED besagt, daß in der Datei Kennsätze (Etiketten) fehlen; STANDARD hingegen verweist auf standardisierte Band- bzw. Plattenkennsätze. OMITTED muß für Dateien, die Satzeinheitengeräten (siehe Abschnitt 17.8.) zugeordnet sind, unbedingt angegeben werden. STANDARD muß für Dateien erscheinen, die externen Speichergeräten mit direkter Zugriffsmöglichkeit zugeordnet sind.

Durch die Klausel DATA sind die Datennamen dn1, dn2 usw. derjenigen Sätze zu erfassen, die zur Datei gehören. Auf jeden FD-Satz müssen die Beschreibungen aller zur Datei gehörenden Sätze folgen. Aus diesen Beschreibungen entnimmt der Kompilierer die Informationen für die Klausel DATA. Er selbst betrachtet nämlich die niedergeschriebene DATA-Klausel nur als Kommentar.

Für jede im Maschinenteil genannte Datei muß ein FD-Satz niedergeschrieben werden.

17.10. Die Datenbeschreibung des Datenteils

Daten, die nicht weiter unterteilt sind, heißen Elemente. Eine benannte Elementenfolge heißt Gruppe. Eine Gruppe kann sowohl Elemente als auch andere Gruppen enthalten. Um die Struktur eines Satzes auszudrücken, werden in COBOL Stufennummern benutzt. Die Stufennummer Ø1 wird jedem Satz zugeordnet. Innerhalb des Satzes muß jedem Teil eine höhere Stufennummer zugeordnet sein, z.B. Ø7. Ist ein Satzteil weiter unterteilt, so müssen die Unterteilungen höhere Stufennummern als der unterteilte Satzteil erhalten, hier z.B. etwa Ø8. Dieser Prozeß der Zuordnung von Stufennummern muß solange fortgesetzt werden, bis nur noch Elemente übrig bleiben. Nur Stufennummern im Wertebereich von Ø1 bis einschließlich 50 können zur Beschreibung eines Satzes dienen. Die spezielle Stufennummer 77 wird im Arbeitsspeicherkapitel zur Kenntlichmachung von Elementen verwendet, die nicht Teil irgendeines Satzes sind. Stufennummern, die kleiner als 10 sind, können auch einstellig geschrieben werden, d.h. 1,7,8 usw.

Für jede Gruppe lautet die Eintragung:

$$\underline{\text{stn}} \left\{ \begin{matrix} \underline{\text{datenname}} \\ \underline{\text{FILLER}} \end{matrix} \right\}$$

Das reservierte Wort FILLER kann an Stelle des Datennamens immer dann benutzt werden, wenn im Programm keine explizite Bezugnahme auf die betreffende Gruppe erfolgt. Unter stn ist die Stufennummer aufzuführen.

Für jedes Datenelement ist folgende Erklärung niederzuschreiben:

$$\underline{\text{stn}} \left\{ \begin{matrix} \underline{\text{datenname}} \\ \underline{\text{FILLER}} \end{matrix} \right\} \; [[\underline{\text{USAGE}} \; \text{IS}] \; \underline{\text{spf}}] \; \underline{\text{PICTURE}} \; \text{IS} \; \underline{\text{abb}}$$

Die einzelnen Teile dieser Erklärung können in einer beliebigen Reihenfolge erscheinen; Stufennummer stn und Datenname datenname bzw. FILLER müssen jedoch am Anfang stehen.

Die Verwendungsklausel (USAGE-Klausel) besagt, in welcher Form das Datenelement gespeichert bzw. zu speichern ist. Unter spf kann eine der folgenden Speicherungsformen genannt werden:

a) DISPLAY
Jedes Zeichen des Datenelementes belegt eine Stelle (ein Byte), deshalb auch Zeichenform genannt.

b) COMPUTATIONAL
Das Datenelement ist als Dualzahl in Festpunktdarstellung anzusehen. Die erforderlichen Speicherplätze hängen von der Anzahl der Ziffernstellen ab, die in der Abbildung abb genannt sind. Es gilt dabei:

Anzahl der Ziffernstellen	Speicherbedarf
1 bis 4	Halbwort
5 bis 9	Vollwort
10 bis 18	Zwei Vollwörter (Doppelwort)

c) COMPUTATIONAL-3
Das Datenelement wird als Dezimalzahl in gepackter Darstellung angesehen. Die erforderlichen Speicherplätze hängen von den unter abb aufgeführten Ziffernstellen n ab. Für die Anzahl a der erforderlichen Bytes gilt

a = (n+1)/2 bei ungeradem n
a = (n+2)/2 bei geradem n

An Stelle von COMPUTATIONAL kann die Abkürzung COMP und an Stelle von COMPUTATIONAL-3 die Abkürzung COMP-3 benutzt werden. Fehlt die USAGE-Klausel, so wird für das betreffende Datenelement die Speicherungsform DISPLAY unterstellt.

Durch die Klausel PICTURE wird die Anzahl der Zeichen bzw. Ziffern festgehalten, aus denen sich das betreffende Datenelement zusammensetzt. Hinzu kommen,natürlich nur bei geeigneten Elementen, noch die Markierung der Stellung des Dezimalpunktes und die Signalisierung der Anwesenheit eines Vorzeichens. Eine vollständige Diskussion über die unter abb aufführbaren Abbildungszeichen folgt später; jetzt begnügen wir uns nur mit den wichtigsten Abbildungszeichen.

Für Datenelemente, die sich aus alphamerischen Zeichen zusammensetzen, muß unter der USAGE-Klausel DISPLAY stehen und die Abbildung abb besitzt die Form

X[X]···

Die Anzahl der Abbildungszeichen X ergibt die Länge des Datenelementes. Um Schreibarbeit einzusprachen, kann ein Wiederholungsfaktor benutzt werden. Dieser ist in Klammern einzuschließen und besagt, daß das vorausgehende Abbildungszeichen zu wiederholen ist. Äquivalente Bedeutung haben infolgedessen:

```
XXXXXX
X(6)
XX(4)X
```

Für numerische Datenelemente gelten die folgenden Abbildungszeichen:

9	Festlegung einer Dezimalziffernstelle
V	Festlegung der Dezimalpunktstellung
S	Festlegung der Anwesenheit eines Vorzeichens

In einer Abbildung abb kann höchstens ein Abbildungszeichen V und ein Abbildungszeichen S erscheinen. Das Nichtauftreten des Zeichens S besagt, daß die Zahl ohne Vorzeichen vorliegt. Vorzeichenfreie Zahlen werden bei der Ausführung von Rechenoperationen immer als positiv angesehen. Das Nichtvorhandensein des Abbildungszeichens V bedeutet, daß der Dezimalpunkt rechts von der Ziffernstelle niedrigster Ordnung angenommen wird, d.h. die Zahl wird als ganze Zahl betrachtet. Das Abbildungszeichen 9 dient zur Markierung jeder Ziffernstelle einer Zahl; die Benutzung von Wiederholungsfaktoren ist dabei gestattet.

Das Auftreten von V und S in einer Abbildung führt nicht zur Reservierung von zusätzlichem Speicherplatz für die betreffende Zahl; beide Abbildungszeichen dienen nur zur näheren Charakterisierung der Zahl. Für Datenelemente in der Speicherungsform DISPLAY bewirkt das S eine Vorzeichenmarkierung („Überlochung") auf der Einerstelle der Zahl, d.h. auf dem am weitesten rechts gelegenen (rechtsbündigen) Byte der Zahl. Wird das Abbildungszeichen S benutzt, so muß es das erste Abbildungszeichen in der Abbildung abb sein.

Die Stufennummern 01 und 77 müssen innerhalb der Randbegrenzung A beginnen; alle anderen Stufennummern können entweder innerhalb der Randbegrenzung A oder innerhalb der Randbegrenzung B anfangen. Der Rest des Satzes mit der Datenbeschreibung muß jedoch gänzlich auf den Stellen untergebracht sein, die auf den einzelnen Zeilen nach links durch die Randbegrenzung B begrenzt sind. Jeder Beschreibungssatz muß mit einem Punkt enden.

Wenn in einer Datei Sätze mehrerer Satzarten vorliegen, so folgt eine Satzbeschreibung auf die andere; der FD-Satz geht selbstverständlich der ersten Satzbeschreibung voraus.

FD-Eintragungen dürfen im Arbeitsspeicherkapitel nicht auftreten. Wenn in diesem Kapitel Elemente mit der Stufennummer 77 definiert werden, so müssen diese Definitionen vor den ersten Eintragungen mit der Stufennummer 01 erscheinen.

Eine dritte Klausel ist nur für Datenelemente im Arbeitsspeicherkapitel erlaubt. Sie lautet:

VALUE IS literal

Wird diese Klausel benutzt, so wird der unter literal aufgeführte Wert dem Datenelement zu Programmbeginn zugewiesen. Ein solcher Wert heißt deshalb auch Anfangswert. Während der Programmausführung kann durch den Programmierer eine Wertänderung veranlaßt werden, genau so, wie er sie bei den Datenelementen ohne die VALUE-Klausel vornehmen kann. An Stelle der Literale können auch figurative Konstanten niedergeschrieben werden.

In der Abb. 17.1. (siehe Abschnitt 17.2.) finden wir einen kurzen, doch typischen Datenteil.

17.11. Der Prozedurteil

Der Prozedurteil beginnt stets mit

PROCEDURE DIVISION.

Das Wort PROCEDURE muß dabei innerhalb der Randbegrenzung A anfangen. Der Rest des Prozedurteils besteht aus einzelnen Sätzen, durch die die vorgesehene Verarbeitung festgelegt ist, sowie aus Paragraphennamen, durch die bestimmte Stellen innerhalb des Prozedurteils eindeutig identifiziert werden können. Paragraphennamen begin-

nen an der Randbegrenzung A; die Sätze müssen vollständig innerhalb der durch die Randbegrenzung B begrenzten Stellen der Zeilen liegen.

Jedes nichtreservierte Wort darf als Paragraphenname verwendet werden. Dem Namen selbst muß ein Punkt folgen. Ein erstes Beispiel hierzu zeigt uns die Abb. 17.7.

```
P.   READ D  AT END GO TO PEND.
     ADD SUMME TO GESAMT-SUMME.
     GO TO P.
PEND.
     CLOSE D.
```

Abb. 17.7. Beispiel für den Aufbau des Prozedurteils

Sowohl P als auch PEND sind Paragraphennamen. Der erste Satz eines Paragraphen kann auf der gleichen Zeile anfangen, auf der der Paragraphenname notiert ist, aber auch auf der folgenden Zeile. Das Quellenprogramm ist sicher leichter zu lesen, wenn jeder Satz auf einer neuen Zeile beginnt.

In diesem Abschnitt wollen wir ein paar grundlegende Statements betrachten, mit deren Hilfe wir bereits sehr einfache Programme abfassen können.

Um den Wert eines Datenelementes von einem Feld in ein anderes zu übertragen, verwenden wir den Satz

MOVE elem TO dn.

Hierbei bedeutet dn einen Datennamen, und elem ist entweder ein Datenname, ein Literal oder eine figurative Konstante. Der Wert von elem wird durch die Ausführung dieses Statements dn zugewiesen. Sind sowohl elem als auch dn numerischer Natur, so erfolgt eine numerische Übertragung, anderenfalls eine nichtnumerische. Bei einer numerischen Übertragung wird der übertragene Wert der Stellung des Dezimalpunktes angepaßt und so umgewandelt, wie es die USAGE- und PICTURE-Angaben verlangen. Bei einer nichtnumerischen Übertragung hingegen werden die einzelnen Zeichen von links nach rechts übertragen, wobei erforderlichenfalls entweder das Zielfeld mit Leerzeichen [1] aufgefüllt wird oder aber die restlichen zu übertragenden Zeichen nach Erreichung des Zielfeldendes abgeschnitten werden. Zur Illustration verweisen wir auf das Beispiel in Abb. 17.8.

```
     .
     .
     .
WORKING-STORAGE SECTION.
Ø1   A.
 Ø2  B PICTURE 99V99 VALUE 68.34.
 Ø2  C PICTURE 99V9 VALUE 17.6 COMPUTATIONAL-3.
 Ø2  D.
  Ø3 E PICTURE XX.
  Ø3 F PICTURE 999 VALUE 2.
PROCEDURE DIVISION.
     MOVE C TO B.
     MOVE C TO F.
     MOVE C TO E.
     MOVE C TO D.
     .
     .
     .
```

Abb. 17.8. Beispiel für Übertragungen von Werten

1) Leerzeichen gelten also bei nichtnumerischen Übertragungen als Basiszeichen.

Als Resultate ergeben sich bei den in Abb. 17.8. vollzogenen Übertragungen folgende Werte:

Zielfeld	Wert
B	17.6Ø
F	Ø17
E	17 6F (sedezimal)
D	17 6F 4Ø 4Ø 4Ø (sedezimal)

Die Verarbeitung von Dateien wird durch das Statement [1]

```
OPEN [INPUT {dateiname} ···]
     [OUTPUT {dateiname} ··· .]
```

eingeleitet. Unter dateiname sind diejenigen Namen aufzuführen, die in den FD-Eintragungen des Datenteils definiert sind. Durch ein einziges OPEN-Statement kann eine beliebige Anzahl von Dateien eröffnet werden. Dabei spielt es keine Rolle, ob sie für Eingabezwecke (INPUT) oder für Ausgabezwecke (OUTPUT) bestimmt sind; eines dieser beiden Wörter muß jedoch jeweils als erstes niedergeschrieben werden. Auf ein Datenelement, das im Dateienkapitel erklärt ist, kann erst dann bezug genommen werden, wenn die Datei, zu der es gehört, eröffnet ist.

Bei einer Eingabedatei wird der nächste logische Satz durch das Statement

```
READ dateiname RECORD [INTO satzname]
     AT END statement.
```

bereitgestellt. Unter dateiname ist der Name der eröffneten Datei aufzuführen. Unter satzname ist eine Eintragung mit der Stufennummer 01 im Arbeitsspeicherkapitel zu verstehen. Schließlich bedeutet statement ein beliebiges Statement oder eine Gruppe von Statements (dann nämlich, wenn die Punkte am Ende aller Statements mit Ausnahme des letzten Statements weggelassen werden). Zur Ausführung des bzw. der auf diese Weise festgelegten Statements kommt es beim ersten READ-Statement, das auf dasjenige folgt, das den letzten Satz der betreffenden Datei bereitgestellt hat. Wird die INTO-Klausel niedergeschrieben, so ist der Effekt der gleiche, als wenn wir die Statementfolge

```
READ dateiname  AT END statement.
MOVE dn TO satzname.
```

codiert hätten. Hierbei ist unter dn der Name eines Satzes zu verstehen, der zur Datei mit dem Dateinamen dateiname gehört, d.h. einer Eintragung mit der Stufennummer 01 im Dateienkapitel. Auf ein Datenelement innerhalb des Dateienkapitels kann erst dann bezug genommen werden, wenn ein READ-Statement für die betreffende Datei ausgeführt wurde. Jede Bezugnahme hat jedoch nach dem Auftreten der Bedingung AT END zu unterbleiben.

Bei einer Ausgabedatei bewirkt die Niederschrift von

```
WRITE satzname [FROM dn].
```

die folgende Operation: In die angesprochene Datei wird der nächste Satz eingefügt. Als angesprochene Datei gilt hierbei diejenige, zu der im Dateienkapitel der Satz mit dem Namen satzname gehört, d.h. satzname muß eine Eintragung mit der Stufennummer 01

1) Ein Satz des Prozedurteils kann aus ein oder mehreren Anweisungen bestehen. Es wird durch einen Punkt abgeschlossen. Statement ist ein Sammelbegriff für Anweisungen und Vereinbarungen.

sein. Es ist zu beachten, daß bei Eingabestatements der Dateiname, bei Ausgabestatements dagegen der Name eines Datensatzes erscheint. (Bei Sätzen, die zu den Satzarten V und U gehören, kann das einen Unterschied zur Folge haben.) Wird bei einem Ausgabestatement die Klausel FROM verwendet, so ist das WRITE-Statement gleichwertig der Statementfolge

```
MOVE dn TO satzname.
WRITE satzname.
```

Vor dem Programmende muß jede eröffnete Datei durch das Statement

```
CLOSE {dateiname} ....
```

abgeschlossen werden. Nach dem Abschließen einer Datei ist die Bezugnahme auf irgendein zu ihr gehörendes Datenelement im Dateienkapitel ungültig.

Der Satz

```
GO TO paragraphenname.
```

überführt die Programmablaufsteuerung zum ersten Satz, der auf den angegebenen Paragraphennamen folgt.

Jedes in COBOL geschriebene Programm kommt zum Abschluß, wenn dem Statement

```
STOP RUN.
```

begegnet wird. Die Programmablaufsteuerung wird dabei an das Betriebssystem zurückgegeben, wobei ein normales Step- bzw. Jobende signalisiert wird.

Betrachten wir nunmehr die Statements, die die Ausführung arithmetischer Operationen zur Folge haben. Das Statement

```
ADD elem TO dn.
```

bewirkt, daß der Wert von dn ersetzt wird durch die Summe aus dn und elem. Das Statement

```
SUBTRACT elem FROM dn.
```

bewirkt eine Subtraktion, und zwar wird elem von dn abgezogen, wobei die Differenz wiederum dn zugewiesen wird. Das Statement

```
MULTIPLY elem BY dn.
```

ersetzt den Wert von dn durch das Produkt aus dn und elem. In gleicher Weise führt das Statement

```
DIVIDE elem INTO dn.
```

dazu, daß der Quotient aus dn und elem, d.h. dn/elem, den Wert von dn ersetzt. Bei allen hier erwähnten arithmetischen Statements steht elem für ein Literal oder eine figurative Konstante oder einen Datennamen, dn hingegen stets für einen Datennamen.

Der Satz

```
IF bed THEN st-1 [ELSE st-2].
```

bewirkt zunächst die Ermittlung des Wertes der unter bed genannten Bedingung. Führt diese zur Aussage „wahr", so wird das Statement st-1 ausgeführt und st-2 wird übergangen; bei „falsch" wird dagegen st-2 angezogen und st-1 außer acht gelassen. Fehlt die ELSE-Klausel, so kommt es bei „falsch" zu keiner Operation. Die einfachste Form einer Bedingung besitzt das folgende Aussehen:

```
a-1 operator a-2
```

Hierbei bedeuten a-1 und a-2 Literale, figurative Konstanten, Datennamen oder arithmetische Ausdrücke. Als Operator kann einer der nachstehenden eingesetzt werden:

```
IS [NOT] GREATER THAN
IS [NOT] >
IS [NOT] LESS THAN
IS [NOT] <
IS [NOT] EQUAL TO
IS [NOT] =
```

Wir wollen die Betrachtungen über das IF-Statement durch ein typisches Beispiel abrunden:

```
IF  A>4.72  MOVE 3.18 TO B.
```

Man beachte, daß ein Punkt erst am Ende des Statements (Satzes) niederzuschreiben ist und nicht bereits am Ende der THEN-Klausel.

Beispiele für die in diesem Abschnitt behandelten Statements mit Ausnahme der arithmetischen finden wir in der Abb. 17.1. (siehe Abschnitt 17.2.).

17.12. Über das Schreiben von COBOL-Programmen

Wir haben genügend Kenntnisse über COBOL gewonnen, um simple Programme schreiben zu können. Die Frage ist nur die, wie wir am besten beginnen.

Angenommen, die Programmablaufpläne sind bereits gewissenhaft überprüft. Man beginnt dann sinnvollerweise mit der Betrachtung der Dateien. Für alle Satzarten jeder Datei ist zunächst ein Entwurf anzufertigen. Die Entwürfe münden schließlich in die Niederschrift des Dateienkapitels des Datenteils. Die Programmablaufpläne zeigen auch, daß einige Hilfsgrößen erforderlich sind, die wiederum in das Arbeitsspeicherkapitel des Datenteils aufzunehmen sind.

Bei der Auswahl der Datennamen muß man darauf achten, daß man nicht versehentlich reservierte Wörter benutzt. Außerdem müssen die Datennamen eindeutig sein.

Im Anschluß daran wird man sich um die Niederschrift des Erkennungsteils und des Maschinenteils bemühen.

Auf den Anfänger kommen bei dieser Vorgehensweise zwei Probleme zu. Beim Schreiben des Datenteils ist einmal zu entscheiden, wo und in welcher Weise Datengruppen gebildet werden sollen, und außerdem, welche Namen gewählt werden sollen. Im Zweifelsfall sollte man für den Namen plädieren, der den Inhalt eines Feldes kennzeichnet. Bei mehreren Sätzen mit gleichen Feldern sollte durch eine Vorsilbe oder eine Nachsilbe für eine Unterscheidung gesorgt werden. Wenn beispielsweise in Sätzen der Dateien HD und GD die gleichen Felder auftreten, so könnte man diese etwa mit HD-ID-CODE und mit GD-ID-CODE bezeichnen. Eine Gruppe sollte immer dann eingeführt werden, wenn mehrere logisch zusammengehörende bzw. zueinander in Beziehung stehende Datenelemente vorliegen. Erinnern wir uns: Es passiert nichts, wenn wir einen unbenutzten Namen im Programm stehen haben. Abgesehen davon läßt er sich später leicht eliminieren.

Der Prozedurteil wird als letzter abgefaßt. Bei seiner Niederschrift wird es sich oft nicht umgehen lassen, im Bedarfsfall laufend das Arbeitsspeicherkapitel des Datenteils zu ergänzen.

Ist das erste Programm verfaßt, so ist es anschließend auf Lochkarten zu übertragen. Die abgelochten Karten sind dem Computer über ein Lochkartenlesegerät zuzuführen. Dabei ist es notwendig, sie durch Steuerstatements der Jobsteuersprache einzurahmen. Diese sind auf das eingesetzte Betriebssystem abzustimmen (siehe Abb. 17.9.).

Betriebssystem OS

```
//jobname JOB MSGLEVEL=1
//stepname EXEC COBUC,PARM='NODECK'
//COB.SYSIN DD *
  (Quellenprogramm)
/*
```

Betriebssystem DOS

```
// JOB jobname
// EXEC FCOBOL
  (Quellenprogramm)
/*
/&
```

Abb. 17.9. Steuerstatements zum Übersetzen von in COBOL geschriebenen Programmen (Übersicht)

Nach erfolgreicher Übersetzung ist das Programm auszuführen. Hierzu sind Kartenpakete zusammenzustellen, die betriebssystemunabhängig sind; in Abb. 17.10. sind sie aufgezeigt.

Betriebssystem OS

```
//jobname JOB MSGLEVEL=1
//stepname EXEC COBUCG
//COB.SYSIN DD *
  (Quellenprogramm)
/*
//GO.ddname DD ···   } erforderliches
:                    } DD-Statement
:
```

Betriebssystem DOS

```
// JOB jobname
// OPTION LINK
// EXEC FCOBOL
  (Quellenprogramm)
/*
  (erforderliches LBLTYP-Statement)
// EXEC LNKEDT
  (erforderliche ASSGN- und LBLTYP-Statements)
// EXEC
  (Datenkarten -wenn vorgesehen-)
/*
/&
```

Abb. 17.10. Steuerstatements für die Ausführung von Programmen (Übersicht)

Hinsichtlich der Jobnamen und der Stepnamen können irgendwelche geeigneten Namen verwendet werden, u.a. auch der entsprechend beschnittene Familienname des Programmierers.

Aufgabengruppe 45

1) Reihe die folgenden Wörter in eine der nachstehend aufgeführten Klassen ein:
   - Gültiger COBOL-Datenname
   - Gültiges COBOL-Wort, jedoch kein Datenname
   - Ungültiges COBOL-Wort

   a) EMPL NO
   b) -A2 1
   c) WORK-AREA-1-
   d) HOLD-AREA. 2
   e) RECORD-123
   f) 1765432
   g) Ø2Ø12Ø
   h) OS I ZUV
   i) 98-78
   j) 795CHR

2) Welche der nachstehend aufgeführten Begriffe sind innerhalb der Randbegrenzung A zu beginnen?

a) Die Namen von Programmteilen
b) Die Datennamen bei Datensatzerklärungen
c) Die Stufennummer Ø3 vor einem Datennamen in einer Datensatzerklärung
d) Der Name des Programmes im Paragraphen PROGRAM-ID
e) Die SELECT-Sätze

3) Welche der nachstehend aufgeführten Zeichenfolgen kann als Datenname Verwendung finden?

a) ENDOFJOB
b) .Ø25
c) ZEROS
d) DF+3
e) A-B-C

4) Reihe die nachstehend aufgeführten Zeichenfolgen in eine der folgenden Klassen ein:

- Nichtnumerisches Literal
- Numerisches Festpunktliteral
- Numerisches Gleitpunktliteral
- Kein COBOL-Literal

a) 1Ø25
b) -76.32
c) 2.4E-7
d) 'SPECKLI'
e) HJKLM
f) HU'JK
g) 'HU'JK'
h) 'H 78'
i) 'KT 9Q'

5) Die nachstehend aufgeführten Datenerklärungen sind einer der folgenden Klassen zuzuordnen!

(1) Gruppe
(2) Alphamerisches Datenelement
(3) Numerisches Datenelement mit dualer Basis
(4) Numerisches Datenelement in gepackter Dezimaldarstellung
(5) Numerisches Datenelement in ungepackter Darstellung (Zeichenform)

Es liegen folgende Erklärungen vor:

a) Ø1 AHUPT-EINGABE-SATZ.
b) Ø2 FILLER PICTURE X(4).
c) Ø3 AUDI-8Ø COMPUTATIONAL-3 PICTURE S9V99.
d) Ø4 BK-35Ø PICTURE S9V99.
e) Ø5 BMW PICTURE S9(5) COMPUTATIONAL-3.
f) Ø6 FILLER PICTURE S999 COMPUTATIONAL.
g) Ø7 DETAIL.
h) Ø8 OPEL PICTURE XXXX.
i) Ø9 TRAFALGAR PICTURE 999 USAGE DISPLAY.
j) 1Ø CAMPOAMOR.

6) Schreibe in COBOL ein Programm zum Lesen der Karten eines Kartenpaketes. Die Karteninhalte sind anschließend zu drucken, wobei jeweils nach der 8., 16., 24., 32. usw. Kartenspalte ein Leerzeichen erscheinen soll.

7) Schreibe in COBOL ein Programm, das Lochkarten einliest, mit den Inhalten bestimmter Kartenspalten Rechenoperationen ausführt und deren Ergebnisse zusammen mit den Eingabewerten druckt. Folgende Voraussetzungen gelten:

   a) Der Inhalt der Kartenspalten 1 bis 8 ist X zuzuweisen (ganze Zahl mit Vorzeichen).
   b) Der Inhalt der Kartenspalten 9 bis 16 ist Y zuzuweisen (ganze Zahl mit Vorzeichen).
   c) Es ist Z nach der Formel

      Z = 2*X + 3*Y

      zu berechnen.
   d) Im Druckbild sollen zwischen den Werten von X, Y und Z je fünf Leerstellen bleiben; für den Wert von Z sind neun Druckstellen zu reservieren.

17.13. Alphabetische Daten

Der Rest dieses Kapitels befaßt sich mit den USAGE- und PICTURE-Klauseln im Datenteil sowie dem Übertragungsstatement MOVE und den arithmetischen Statements im Prozedurteil.

Unter einem alphabetischen Datenelement versteht man ein Datenelement, das ausschließlich aus den 26 Buchstaben und dem Leerzeichen gebildet ist. In der USAGE-Klausel darf nur DISPLAY angegeben sein. Als Abbildungszeichen in der PICTURE-Klausel darf nur das Zeichen A auftreten. Die Anzahl der Abbildungszeichen A gibt Auskunft über die vom Datenelement beanspruchten Bytes.

Die Definition eines alphabetischen Datenelementes beinhaltet nicht, daß das Element immer einen Wert annimmt, der nur aus alphabetischen Zeichen besteht. Sie besagt vielmehr dem Kompilierer, daß der Programmierer sichergehen will, daß das Element niemals einen anderen als einen alphabetischen Wert enthalten soll, wenn das Programm auf dieses Element zugreift.

Bei der Übertragung eines alphabetischen Datenelementes zu einem anderen alphabetischen Datenelement wird bei COBOL ein nichtnumerischer Transport veranlaßt. Die Übertragung beginnt demnach beim linksbündigen Byte jedes Elementes und schreitet Byte für Byte von links nach rechts fort. Ist das Empfangsfeld kleiner als das Sendefeld, so werden die am weitesten rechts gelegenen Zeichen des Sendefeldes nicht mehr bewegt. Ist dagegen das Empfangsfeld länger, so werden Leerzeichen in die verbleibenden Bytes des Empfangsfeldes übertragen.

Abb. 17.11. zeigt einige typische MOVE-Statements mit alphabetischen Daten und ihre Ergebnisse.

Datenteil

```
77 AD1   PICTURE A(3)   VALUE 'KLM'.
77 AD2   PICTURE A(4)   VALUE 'STUV'.
77 AD3   PICTURE A(4).
77 AD4   PICTURE A(3).
```

Prozedurteil

Statements	Resultate
MOVE AD1 TO AD3.	AD3 ⟵ 'KLMb'
MOVE AD2 TO AD4.	AD4 ⟵ 'STU'

Anmerkung: b bedeutet hier das Leerzeichen

Abb. 17.11. Beispiel von Übertragungen mit alphabetischen Daten

17.14. Alphamerische Daten

Unter einem alphamerischen Datenelement [1] wird ein Datenelement verstanden, in dessen Wertevorrat jedes der möglichen 256 Zeichen des EBCDI-Codes eingeschlossen ist. Die USAGE-Klausel muß DISPLAY beinhalten. In der PICTURE-Klausel dürfen nur die Abbildungszeichen A, X und 9 erscheinen. Jedes Abbildungszeichen A verweist auf eine Stelle des Datenelementes, die ein alphabetisches Zeichen (Buchstabe oder Leerzeichen) aufnehmen kann. Durch das Abbildungszeichen X ist eine Stelle des Datenelementes repräsentiert, die irgendein Zeichen des EBCDI-Codes aufnimmt und schließlich verweist das Abbildungszeichen 9 auf eine Stelle, die nur Dezimalziffern aufnehmen kann. Im allgemeinen wird bei einer alphamerischen Abbildung nur das Abbildungszeichen X benutzt. Die Voraussetzung für eine alphamerische Abbildung ist dann erfüllt, wenn mindestens ein Abbildungszeichen X vorkommt oder sowohl A als auch 9. Die Gesamtzahl der Abbildungszeichen ist gleich der vom Datenelement belegten Bytes.

Alphamerische Datenelemente werden infolge ihrer Universalität beinahe immer an Stelle von alphabetischen Datenelementen benutzt. Es gibt allerdings einige wenige Anwendungen in der Datenverarbeitung, bei denen der Programmierer aus der Tatsache Vorteile ziehen kann, daß ein bestimmtes Datenelement nur aus alphabetischen Zeichen besteht, jedoch keine Dezimalziffern oder Sonderzeichen umfassen darf. Unberücksichtigt der in der Abbildung aufgeführten Abbildungszeichen betrachten die Kompilierer der DV-Systeme IBM /360 und IBM /370 die Abbildungen so, als ob sie nur aus den Abbildungszeichen X bestehen würden.

Datenteil

```
77 AD1   PICTURE A(3)   VALUE 'KLM'.
77 AD2   PICTURE A(4)   VALUE 'STUV'.
77 AD3   PICTURE A(4).
77 AD4   PICTURE A(3).
77 AE1   PICTURE X(3)   VALUE '#3A'.
77 AE2   PICTURE X(5)   VALUE '45ØSL'.
77 AE3   PICTURE X(5).
77 AE4   PICTURE X(3).
```

Prozedurteil

Statements	Resultate
MOVE AD1 TO AE3.	AE3 ⟵ 'KLMbb'
MOVE AD2 TO AE4.	AE4 ⟵ 'STU'
MOVE AE1 TO AE3.	AE3 ⟵ '#3Abb'
MOVE AE2 TO AE4.	AE4 ⟵ '45Ø'
MOVE AE1 TO AD3.	AD3 ⟵ '#3Ab'
MOVE AE2 TO AD4.	AD4 ⟵ '45Ø'

Abb. 17.12. Beispiele von Übertragungen mit alphamerischen Daten

Anmerkung: b bedeutet hier das Leerzeichen

1) Mitunter auch als alphanumerische Daten bezeichnet.

In der Abb. 17.12. werden Beispiele für das MOVE-Statement mit alphamerischen Datenelementen gezeigt. Selbstverständlich wird durch dieses Statement wie bei den alphabetischen Datenelementen ein nichtnumerischer Transport veranlaßt. Eine Übertragung zwischen alphabetischen und alphamerischen Datenelementen ist gestattet. Ein in COBOL geschriebenes Programm führt zu keiner Überprüfung in der Hinsicht, ob nur alphabetische Zeichen zu alphabetischen Datenelementen übertragen werden. Bei der Abb. 17.12. verweisen wir besonders darauf, daß die beiden letzten MOVE-Statements nichtalphabetische Daten zu alphabetischen Datenelementen übertragen.

17.15. Aufbereitete alphamerische Daten

Unter einem aufbereiteten alphamerischen Datenelement [1] versteht man ein Datenelement, das unter Einsatz der Abbildungszeichen A, X, 9, B und 0 definiert ist. Dabei muß die Abbildung mindestens einer der folgenden Voraussetzungen genügen:

1) Es muß mindestens ein Abbildungszeichen A und ein Abbildungszeichen 0 vorkommen.

2) Es muß mindestens ein Abbildungszeichen X und ein Abbildungszeichen 0 vorkommen.

3) Es muß mindestens ein Abbildungszeichen X und ein Abbildungszeichen B vorkommen.

Die USAGE-Klausel muß DISPLAY beinhalten. Jedes Abbildungszeichen repräsentiert den Inhalt eines Bytes, wobei den einzelnen Zeichen die folgende Bedeutung zukommt:

Abbildungs-zeichen	Bedeutung
A	Irgendein Buchstabe oder ein Leerzeichen
X	Irgendein Zeichen des EBCDI-Codes
9	Irgendeine Dezimalziffer
B	Leerzeichen
0	Ziffer 0

Wird für ein aufbereitetes alphamerisches Datenelement die VALUE-Klausel benutzt, so muß der Wert als nichtnumerisches Literal (siehe Abschnitt 17.4.) aufgeführt sein. Der bei VALUE angegebene Wert wird dem Datenelement genau so zugewiesen, wie er niedergeschrieben ist, sogar dann, wenn er nicht der festgelegten Abbildung Genüge leistet.

Wenn der Wert eines aufbereiteten alphamerischen Datenelementes einem alphabetischen oder einem alphamenischem Datenelement zugewiesen wird, so wird das Sendefeld als ein Feld betrachtet, dessen Abbildung aus lauter X besteht; infolgedessen erfolgt eine nichtnumerische Übertragung. Wird hingegen der Wert eines alphabetischen, alphamerischen oder aufbereiteten alphamerischen Datenelement zugewiesen, so findet die Aufbereitung zusammen mit der Übertragung vom Sendefeld zum Empfangsfeld statt. Jedes Abbildungszeichen A, X und 9 wird dabei vom Inhalt eines Bytes des Sendefeldes ersetzt. An Stelle des Abbildungszeichens X wird ein Leerzeichen und an Stelle des Abbildungszeichens 0 die Ziffer 0 in das Empfangsfeld eingeschoben. In der Abb. 17.13. werden mehrere Übertragungen vorgestellt, in die aufbereitete alphamerische Daten verwickelt sind.

1) Auch als aufbereitete alphanumerische Daten bezeichnet

Datenteil

```
77 AL1   PICTURE A(3)   VALUE 'KLM'.
88 AL2   PICTURE A(4)   VALUE 'STUV'.
77 AL3   PICTURE A(4).
77 AL4   PICTURE A(3).
77 AN1   PICTURE X(3)   VALUE '#3A'.
77 AN2   PICTURE X(5)   VALUE '45ØSL'.
77 AN3   PICTURE X(5).
77 AN4   PICTURE X(3).
77 AE1   PICTURE XØØX   VALUE 'YØØZ'.
77 AE2   PICTURE ØX     VALUE 'ØE'.
77 AE3   PICTURE XØØX.
77 AE4   PICTURE XBXXØ.
```

Prozedurteil

Statements	Resultate
MOVE AL1 TO AE3.	AE3 ←——— 'KØØL'
MOVE AL2 TO AE4.	AE4 ←——— 'SbTUØ'
MOVE AN1 TO AE3.	AE3 ←——— '#ØØ3'
MOVE AN2 TO AE4.	AE4 ←——— '4b5ØØ'
MOVE AE1 TO AE3.	AE3 ←——— 'YØØØ'
MOVE AE2 TO AE4.	AE4 ←——— 'ØbEbØ'
MOVE AE1 TO AL3.	AL3 ←——— 'YØØZ'
MOVE AE2 TO AL4.	AL4 ←——— 'ØEb'
MOVE AE1 TO AN3.	AN3 ←——— 'YØØZ'
MOVE AE2 TO AN4.	AN4 ←——— 'ØEb'

Anmerkung: b bedeutet hier das Leerzeichen

Abb. 17.13. Beispiele von Übertragungen mit aufbereiteten alphamerischen Daten

17.16. Externe Dezimaldaten

Unter einem externen Dezimaldatenelement versteht man ein Datenelement, das nur Dezimalziffern und evtl. ein Vorzeichen enthält. In der USAGE-Klausel muß DISPLAY angegeben sein. In der Abbildung dürfen nur die Abbildungszeichen 9, P, V und S vorkommen. Mindestens einmal muß das Abbildungszeichen 9 vorkommen. Jedes Abbildungszeichen 9 repräsentiert eine Dezimalziffer im Datenelement; jedes P hingegen eine Ziffernstelle, die nicht zum Datenelement gehört. Die Abbildungszeichen P werden benutzt, um den Dezimalpunkt zu lokalisieren. Ist der Wert des Datenelementes mit einem Vorzeichen behaftet, d.h. positiv oder negativ, so muß als erstes Abbildungszeichen das S verwendet werden. Fehlt das S in der Abbildung, so wird das Datenelement als vorzeichenfrei angesehen.

Zusammengefaßt, es sind nur die folgenden Abbildungen erlaubt:

```
[S]9···9[V]
[S]V9···9
[S]9···9V9···9
[S]9···9P···P[V]
[S][V]P···P9···9
```

Bei den drei Abbildungen, in denen die Benutzung des Abbildungszeichens V freigestellt ist, wird bei Fehlen des V es dort angenommen, wo es oben niedergeschrieben ist. Die Gesamtzahl der Abbildungszeichen 9 und P darf 18 nicht überschreiten. Man beachte, daß die Verwendung des Abbildungszeichens P nur dann in Frage kommt, wenn der Dezimalpunkt mindestens eine Ziffernstelle außerhalb des Feldes liegt.

Die Anzahl der vom Datenelement belegten Bytes entspricht der Anzahl der Abbildungszeichen 9 in der Abbildung. Der Zonenteil jedes Bytes enthält die Sedezimalziffer F, ausgenommen jedoch beim am weitesten rechts gelegenen (rechtsbündigen) Byte, das im Zonenteil bei vorzeichenfreien Zahlen die Sedezimalziffer F, bei positiven Zahlen die Sedezimalziffer C und bei negativen Zahlen die Sedezimalziffer D aufweist. Bei einem Zeilendrucker werden die rechtsbündigen Ziffern deshalb wie folgt gedruckt:

A bis I bei +1 bis +9,
J bis R bei -1 bis -9.

Bei +0 und -0 wird überhaupt kein Zeichen gedruckt.

Im Datenelement selbst wird kein Dezimalpunkt gespeichert. Der Kompilierer sorgt dafür, daß die Stellungen der Dezimalpunkte bei Übertragungsoperationen und bei arithmetischen Operationen in korrekter Weise beachtet werden.

Wird der Wert eines externen Dezimaldatenelementes zu einem anderen externen Datenelement übertragen, so erfolgt eine numerische Übertragung: die Ziffern werden rings um den Dezimalpunkt angeordnet. Diese Rundumanordnung kommt folgendem Vorgehen gleich: Das Empfangsfeld wird auf Null gesetzt und anschließend der Inhalt des Sendefeldes auf das Empfangsfeld addiert. Ein Vorzeichen wird dann und nur dann eingefügt, wenn die Abbildung des Empfangsfeldes das Abbildungszeichen S enthält. Ein Abschneiden nach rechts findet dann statt, wenn das Empfangsfeld zu wenig Stellen aufweist, um alle Ziffern des Sendefeldes aufnehmen zu können.

Der Wert eines alphamerischen Datenelementes kann zu einem externen Dezimaldatenelement übertragen werden. Dabei wird das alphamerische Datenelement als ein Datenelement angesehen, dessen Abbildung aus lauter Abbildungszeichen 9 besteht, an das sich das Abbildungszeichen V anschließt. Die Übertragung selbst ist numerisch orientiert.

Die Übertragung von alphabetischen oder aufbereiteten alphamerischen Daten zu einem externen Dezimaldatenelement ist niemals erlaubt. Der Wert eines externen Dezimaldatenelementes kann zu einem alphamerischen oder einem aufbereiteten alphamerischen Datenelement ohne Schwierigkeiten dann übertragen werden, wenn sichergestellt ist, daß das externe Dezimaldatenelement mit einer Abbildung der Form

[S]9···9[V]

erklärt ist. Vor dem Transport wird der Wert so umgewandelt, daß er der Abbildung

9···9[V]

gerecht wird; danach erfolgt erst eine nichtnumerische Übertragung. Ist dagegen dem externen Dezimaldatenelement irgendeine andere Abbildung unterstellt, so ist die Übertragung seines Wertes zu einem alphamerischen Datenelement erlaubt, allerdings wird bei der Ausführung die Stellung des Dezimalpunktes nicht berücksichtigt; die Übertragung zu einem aufbereiteten alphamerischen Datenelement ist jedoch nicht gestattet.

Übertragungen von Werten zwischen externen Dezimaldatenelementen und alphabetischen Datenelementen und umgekehrt sind niemals erlaubt.

In der Abb. 17.14. sind Beispiele für Übertragungen aufgeführt, in die externe Dezimaldaten eingehen.

Datenteil

```
77 AN3  PICTURE X(5).
77 AN4  PICTURE X(3).
77 AN5  PICTURE X(3)  VALUE 'Ø27'.
77 AN6  PICTURE X(4)  VALUE 'Ø796'.
77 AE3  PICTURE XØØX.
77 AE4  PICTURE XBXXØ.
77 ED1  PICTURE S999V99  VALUE -123.65.
77 ED2  PICTURE 9(4)     VALUE 2348.
77 ED3  PICTURE S999V99.
77 ED4  PICTURE 9(4).
77 ED5  PICTURE 999PP.
77 ED6  PICTURE SP99.
```

Prozedurteil

Statements	Resultate		
MOVE AN5 TO ED3.	ED3 ←	FØ F2 F7 FØ CØ	Sedezimaldarstellung
MOVE AN6 TO ED5.	ED5 ←	F7 F9 F6 FØ CØ	
MOVE AN6 TO ED5.	ED5 ←	FØ FØ	
MOVE ED1 TO ED4.	ED4 ←	F1 F2 F3 F6 D5	
MOVE ED1 TO ED4.	ED4 ←	FØ F1 F2 F3	
MOVE ED1 TO ED6.	ED6 ←	F5 DØ	
MOVE ED2 TO ED3.	ED3 ←	F3 F4 F8 FØ CØ	
MOVE ED2 TO ED5.	ED5 ←	FØ F2 F3	
MOVE ED1 TO AN3.	AN3 ←	'12365'	
MOVE ED1 TO AN4.	AN4 ←	'123'	
MOVE ED2 TO AN3.	AN3 ←	'2348'	
MOVE ED2 TO AN4.	AN4 ←	'234'	
MOVE ED2 TO AE3.	AE3 ←	'2ØØ3'	
MOVE ED2 TO AE4.	AE4 ←	'2b34Ø'	

Anmerkung: b bedeutet hier das Leerzeichen

Abb. 17.14. Beispiele von Übertragungen mit externe Dezimaldaten

17.17. Interne Dezimaldaten

Ein internes Dezimaldatenelement entspricht im wesentlichen einem externen Dezimaldatenelement mit der Ausnahme jedoch, daß in der USAGE-Klausel die Eigenschaft COMPUTATIONAL-3 vermerkt sein muß. Der Wert eines solchen Datenelementes wird als gepackte Dezimalzahl gespeichert. Die ihm zukommende Länge ist gleich der geringsten Anzahl von Bytes, die zur Speicherung aller Ziffern benötigt werden. Für ein Datenelement, dessen Abbildung aus m Abbildungszeichen 9 besteht, sind das infolgedessen (m+1)/2 oder (m+2)/2 Bytes, je nachdem, ob m ungerade oder gerade ist.

Da ein für interne Dezimaldatenelemente vorgesehenes Feld stets eine ungerade Anzahl von Ziffernstellen aufweist, wird durch die Kompilierer mitunter ein leistungsfähigerer Maschinencode erzeugt, wenn die Zahl der Abbildungszeichen 9 in der Abbildung ungerade ist. Bei

Datenelementen mit einer geraden Anzahl von Abbildungszeichen 9 in der Abbildung müssen die Kompilierer nämlich einen Maschinencode generieren, der gewährleistet, daß nach arithmetischen Operationen und nach Übertragungsoperationen die Ziffernstelle höchster Ordnung (linksbündige Ziffernstelle) auf Null gesetzt ist.

Wenn der Wert eines Datenelementes, bei dem in der USAGE-Klausel die Eigenschaft DISPLAY vermerkt ist, nach einem Datenelement transportiert wird, bei dem COMPUTATIONAL-3 verzeichnet ist, so wird der Ausgangswert zunächst in ein Feld hineingestellt, das zu einem externen Dezimaldatenelement mit der gleichen Abbildung gehört, die das interne Dezimaldatenelement besitzt. Danach wird das externe Dezimaldatenfeld in das interne umgewandelt, d.h. gepackt. Umgekehrt: Wird der Wert eines internen Dezimaldatenelementes zu einem Datenelement übertragen, bei dem in der USAGE-Klausel die Eigenschaft DISPLAY verzeichnet ist, so wird zunächst der zu übertragende Wert in einen externen Dezimalwert umgewandelt, bevor der Transport erfolgt. In einigen Fällen kann der Kompilierer einen Maschinencode generieren, der die Umwandlung und die Übertragung kombiniert.

17.18. Dualdaten

Ein Dualdatenelement ist mit Ausnahme der Tatsache, daß in der USAGE-Klausel die Eigenschaft COMPUTATIONAL verzeichnet sein muß, identisch einem externen Dezimaldatenelement. Enthält die Abbildung nicht mehr als vier Abbildungszeichen 9, so wird für das Element ein Halbwort bereitgestellt. Weist die Abbildung zwischen 5 und 9 Abbildungszeichen 9 auf, so wird von den Kompilierern ein Wort vorgesehen, bei 10 bis 18 dieser Abbildungszeichen sind es zwei aneinanderstoßende Worte. Vorzeichenfreie Zahlen werden als positive angesehen.

Für die verschiedenen Übertragungsmöglichkeiten gelten folgende Regeln:

a) Ist der Wert eines Dualdatenelementes zu einem Datenelement mit DISPLAY oder COMPUTATIONAL-3 in der USAGE-Klausel übertragen, so wird der zu übertragende Wert zunächst in ein internes Dezimaldatenelement mit der gleichen Abbildung umgewandelt. Der anschließende Transport erfolgt so, als ob er Wert eines internen Dezimaldatenelementes zu übertragen ist.

b) Ist der Wert eines Datenelementes mit DISPLAY oder COMPUTATIONAL-3 in der USAGE-Klausel zu einem Dualdatenelement zu übertragen, so erfolgt zunächst ein Transport zu einem internen Dezimaldatenelement mit der gleichen Abbildung. Der interne Dezimalwert wird anschließend in einen Dualwert umgewandelt.

Um sicherzustellen, daß die Ergebnisse von in COBOL geschriebenen Programmen unabhängig von der in der USAGE-Klausel vermerkten Eigenschaft sind, werden folgende Maßnahmen von den Kompilierern veranlaßt: Immer, wenn mit Dualdaten Rechenoperationen (Additionen usw.) anfallen, wird das Ergebnis in einen internen Dezimalwert umgewandelt, so daß das Abschneiden der Ziffern in den Stellen der höheren und niederen Ordnungen so stattfinden kann, wie es die Abbildung erfordert, die zum Empfangsfeld gehört. Danach wird der interne Dezimalwert wieder in einen Dualwert zurückgebracht. Als Resultat kristallisiert sich bei einem solchen Vorgehen die Tatsache heraus, daß in COBOL-Programmen arithmetische Operationen mit internen Dezimaldaten gewöhnlich schneller ablaufen als solche mit Dualdaten. Allgemein gilt deshalb als Anhaltspunkt, daß interne Dezimaldatenelemente fast immer schneller Ergebnisse zeitigen als Dualdaten oder externe Dezimaldaten.

- Die Versionen 3,4 u.f. der Kompilierer erlauben die Angabe der Eigenschaft COMPUTATIONAL-4 in der USAGE-Klausel; diese ist identisch der Eigenschaft COMPUTATIONAL.

17.19. Aufbereitete numerische Daten

Ein aufbereitetes numerisches Datenelement wird unter Benutzung der folgenden Abbildungszeichen definiert:

9 | P | B | 0 | V | . | , | Z | * | + | - | $ | CR | DB

In einer Abbildung muß mindestens ein Abbildungszeichen ungleich 9, P oder V sein. Die Länge des Datenelementes ergibt sich aus der Anzahl der Abbildungszeichen, wobei freilich die Zeichen P und V nicht zu zählen sind. Dagegen gehen CR und DB als zwei Zeichen in die Längenermittlung ein. Die Maximallänge ist auf 127 Bytes beschränkt; dabei darf jedoch auf nicht mehr als 18 Dezimalzifferstellen verwiesen werden. In der USAGE-Klausel muß stets DISPLAY stehen.

Aufbereitete numerische Daten werden in erster Linie dazu benutzt, um numerische Werte für das anschließende Drucken aufzubereiten. Bei den nachfolgenden Erörterungen sprechen wir von der Übertragung eines numerischen Feldes, wenn wir die Übertragung eines externen Dezimalfeldes oder eines internen Dezimalfeldes oder eines Dualwertes in ein aufbereitetes numerisches Feld meinen. Das numerisch aufbereitete Datenelement selbst ist nicht numerisch.

Jede Abbildung kann ein Zeichen enthalten, das das Auftreten eines Vorzeichens anzeigt. Hierzu bieten sich folgende Möglichkeiten an:

+	Im Empfangsfeld wird ein Byte zur Aufnahme des Vorzeichens (+ bei nichtnegativen Werten) bereitgestellt
-	Im Empfangsfeld wird ein Byte zur Aufnahme des Minuszeichens bei negativen Werten bereitgestellt (bei nichtnegativen Werten erscheint stattdessen ein Leerzeichen)
CR	Die Buchstabenfolge CR erscheint bei negativen Werten, bei nichtnegativen bleiben die betreffenden Stellen leer
DB	Die Buchstabenfolge DB erscheint bei negativen Werten, bei nichtnegativen bleiben die betreffenden Stellen leer

CR und DB können ausschließlich nur als rechtsbündige Abbildungszeichen, d.h. auf den am weitesten rechts gelegenen Stellen der Abbildung, erscheinen; die Abbildungszeichen + bzw. - müssen entweder der linksbündigen oder der rechtsbündigen Stelle der Abbildung vorbehalten bleiben.

Jede Abbildung kann ein Währungszeichen enthalten. Es existiert hierzu die folgende Möglichkeit:

$	Ein Währungszeichen (hier: Dollarzeichen) wird an der so gekennzeichneten Stelle ins Empfangsfeld eingefügt

Dem Währungszeichen kann nur eines der beiden Abbildungszeichen + oder - vorausgehen.

Die Stellung des Dezimalpunktes kann auf drei verschiedene Weisen festgelegt werden:

V	Festlegung der Stellung des Dezimalpunktes, innerhalb des Feldes erscheint jedoch kein Punkt
.	Festlegung der Stellung des Dezimalpunktes, an der betreffenden Stelle wird tatsächlich ein Dezimalpunkt ins Feld eingeschoben

P	Festlegung der Stellung des Dezimalpunktes wie bei numerischen Abbildungen; ein Zeichen selbst wird jedoch durch dieses Abbildungszeichen nicht repräsentiert.

Die beiden Abbildungszeichen . und V dürfen nicht gemeinsam in der gleichen Abbildung auftreten. Fehlen die Abbildungszeichen V, . oder P in einer Abbildung gänzlich, so wird das Abbildungszeichen V rechts vom letzten Abbildungszeichen angenommen, das in der Abbildung eine Ziffernstelle repräsentiert.

Gewisse Abbildungszeichen können innerhalb einer Abbildung an beliebiger Stelle erscheinen. Derartige Zeichen schließen bei der Übertragung die Einfügung eines bestimmten Zeichens an der betreffenden Stelle ein; sie heißen deshalb Einfügungszeichen. Es gibt folgende Einfügungszeichen:

,	Einfügung eines Kommas
B	Einfügung eines Leerzeichens
ø	Einfügung einer Null

Dezimalziffernstellen werden wie bei den numerischen Datenelementen durch die Abbildungszeichen 9 bzw. P repräsentiert.

Die Übertragung eines Wertes in ein aufbereitetes numerisches Feld läuft in zwei Schritten ab: Im ersten Schritt erfolgt die Übertragung desselben in ein numerisches Feld, dessen Abbildung die Abbildungszeichen S und V aufweist sowie alle Abbildungszeichen 9 und P des Empfangsfeldes. Danach wird der auf diese Form gebrachte Wert in das Empfangsfeld unter Beachtung von dessen Abbildung hineingestellt. Ein beliebiger alphamerischer Wert, externer Dezimalwert, interner Dezimalwert oder Dualwert kann einem aufbereiteten numerischen Datenelement zugewiesen werden.

Erscheint bei einem numerisch aufbereiteten Datenelement die VALUE-Klausel, so wird der in ihr spezifizierte Wert, der ein nichtnumerisches Literal sein muß, so benutzt, wie er niedergeschrieben ist; es erfolgt keine Aufbereitung.

Oft ist es wünschenswert, daß führende Nullen durch Leerzeichen ersetzt werden. Es kann sogar der Gedanke auftauchen, ein ganzes Feld mit Leerzeichen zu füllen, wenn sein Wert gleich Null ist. Das Abbildungszeichen Z kann hierzu benutzt werden, wobei folgende Regeln zu beachten sind:

1) Jedes Z repräsentiert eine Ziffernposition. Diese Position wird mit einem Leerzeichen gefüllt, wenn es links von dieser Stelle keine Ziffern ungleich Null und die zu dieser Stelle gehörende Ziffer gleich Null ist. Anderenfalls erscheint die zugehörige Ziffer in dieser Position.

2) Kein Abbildungszeichen Z darf in einer Abbildung rechts von einem der Abbildungszeichen 9 oder P auftreten.

3) Einfügungszeichen (Komma, Leerzeichen und Null), denen nur Leerzeichen, Vorzeichen und Währungszeichen vorausgehen, werden ebenfalls durch Leerzeichen ersetzt.

4) Wird rechts von einer Dezimalpunktposition das Abbildungszeichen Z benutzt, so muß es alle Ziffernstellen rechts vom Dezimalpunkt kennzeichnen. In diesem Fall stoppt die Nullenunterdrückung beim Auftreten eines V oder eines . , es sei denn, daß der zu übertragende Wert gleich Null ist. Dann nämlich werden in das Feld lauter Leerzeichen gebracht. Die Entscheidung darüber, ob ein Datenelement den Wert Null besitzt oder nicht, wird an Hand des Wertes im Zwischenfeld, das weiter oben (1, Schritt der Übertragung) beschrieben ist, ge-

troffen. Für die Nullenunterdrückung ist also nicht der Wert im Sendefeld maßgebend

Die Abbildungszeichen +, - und $ können an Stelle des Abbildungszeichens Z verwendet werden; sie heißen bei einem Gebrauch Gleitzeichen. Nur ein einziges dieser Gleitzeichen (Z, +, -, $) kann in einer Abbildung erscheinen; ein einzelnes Abbildungszeichen für Vorzeichen oder zur Währungskennzeichnung kann jedoch in üblicher Weise zusätzlich benutzt werden. Bei den Gleitzeichen +, - oder $ erfolgt die Einfügung des Vorzeichens oder des Währungszeichens unmittelbar links vor das erste, ein anderes Zeichen als das Leerzeichen enthaltende Byte. Die Anzahl der Ziffernpositionen ist um eins geringer als die Anzahl der angegebenen Gleitzeichen, da eine Stelle mit dem Vorzeichen bzw. Währungszeichen selbst gefüllt werden muß. Die Kompilierer erkennen ein Gleitzeichen daran, daß das betreffende Abbildungszeichen auf zwei aneinanderstoßenden Stellen notiert ist.

Datenteil

```
77 ID1   PICTURE S9999V99   VALUE -1234.56   COMPUTATIONAL-3.
77 ID2   PICTURE 9999        VALUE 8Ø3        COMPUTATIONAL-3.
77 ID3   PICTURE S99V999     VALUE Ø          COMPUTATIONAL-3.
77 ID4   PICTURE SV999       VALUE -.ØØ7      COMPUTATIONAL-3.
```

Prozedurteil

Statements

```
MOVE ID1 TO ···.
MOVE ID2 TO ···.
MOVE ID3 TO ···.
MOVE ID4 TO ···.
```

Resultate

Abbildung des Empfangsfeldes	Wert im Empfangsfeld nach Übertragung vom Sendefeld			
	ID1	ID2	ID3	ID4
Z,ZZZ.99	1,234.56	bb8Ø3.ØØ	bbbbØ.ØØ	bbbbØ.ØØ
$*****.99	$1234.56	$**8Ø3.ØØ	$*****.ØØ	$*****.ØØ
ZBZZZVZZZ	1b23456Ø	bb8Ø3ØØØ	bbbbbbbb	bbbbbØØ7
Z,ZZZ.ZZ	1,234.56	bb8Ø3.ØØ	bbbbbbbb	bbbbbbbb
*,***.**	1,234.56	**8Ø3.ØØ	*****.**	*****.**
,+	**1,234-	****8Ø3+	********	********
$$$,$$$-	b$1,234-	bbb$8Ø3b	bbbbbbbb	bbbbbbbb
99,999CR	Ø1,234CR	ØØ,8Ø3bb	ØØ,ØØØbb	ØØ,ØØØbb
+$999.9Ø	-$234.5Ø	+$8Ø3.ØØ	+$ØØØ.ØØ	+$ØØØ.ØØ
Z,ZZ9,99	1,234.56	bb8Ø3,ØØ	bbbbØ.ØØ	bbbbØ.ØØ
-----.99	-1,234.56	bb8Ø3.ØØ	bbbbb.ØØ	bbbbb.ØØ
$$$$.$$$	$234.56Ø	$8Ø3.ØØØ	bbbbbbbb	bbb$.ØØ7
ZZ,ZØØPP.B	b1,2ØØ.b	bbb8ØØ.b	bbbbbbbb	bbbbbbbb
-99B9PPØØB	-Ø1b2ØØb	bØØb8ØØb	bØØbØØØb	bØØbØØØb
$BBP99BDE	$bb6ØbDB	$bbØØbbb	$bbØØbbb	$bbØ7bDB

Anmerkung: b bedeutet hier das Leerzeichen

Abb. 17.15. Beispiele von Übertragungen zu aufbereiteten numerischen Datenelementen

An Stelle des Abbildungszeichens Z kann auch das Sternzeichen (*) als Abbildungszeichen eingesetzt werden. Es bewirkt, daß die führenden Nullen durch Sternzeichen ersetzt werden. Sind alle Ziffernpositionen des Empfangsfeldes durch Sternzeichen gekennzeichnet und ist der im Sendefeld gespeicherte Wert gleich Null, so wird das Sternzeichen an alle Stellen gebracht; es ersetzt jedoch nicht den Dezimalpunkt.

In der Abb. 17.15. sind eine Reihe von Datenübertragungen dargestellt, deren Ziel aufbereitete numerische Datenelemente sind. Umgekehrt kann der Wert eines aufbereiteten numerischen Datenelementes zu einem alphamerischen oder aufbereiteten alphamerischen Datenelement übertragen werden. Derartige Transporte sind nichtnumerischer Natur. Der Wert eines aufbereiteten numerischen Datenelementes kann niemals einem alphabetischen oder einem numerischen Datenelement zugewiesen werden.

17.20. Externe Gleitpunktdaten

Unter einem externen Gleitpunktdatenelement versteht man ein Element, dessen Wertevorrat aus Zahlen in der Gleitpunktdarstellung, und zwar in der Potenzform, besteht. In der USAGE-Klausel muß DISPLAY vermerkt sein, sofern nicht auf ihre Niederschrift verzichtet wird. Die Form der Abbildung lautet:

mantisse E exponent

Durch Abbildungen dieser Art werden Zahlen repräsentiert, die gleich

$$m \cdot 10^e$$

sind, wobei unter m die Mantisse und unter e der Exponent zu sehen ist.

Für die Mantisse können folgende Abbildungszeichen benutzt werden:

+ | - | 9 | V | .

Wenn + und - benutzt werden, so müssen sie an erster Stelle erscheinen. Die Abbildungszeichen V und . können nicht in der gleichen Abbildung auftreten. Erscheint keines von beiden, so wird das Abbildungszeichen V am rechten Ende der Mantisse angenommen. Die Mantisse kann zwischen 1 und 16 Ziffernstellen enthalten, jede von ihnen ist in der Abbildung durch das Abbildungszeichen 9 zu repräsentieren.

Der Teilabbildung für die Mantisse muß der Buchstabe E folgen. Durch ihn wird das Ende der Mantisse und der Beginn des Exponenten angezeigt. Die Teilabbildung für den Exponenten besitzt entweder die Form

[+] 99

oder die Form

[-] 99

Es müssen stets zwei Abbildungszeichen 9 erscheinen.

Die vom Datenelement belegten Stellen sind bestimmt durch die Anzahl der Abbildungszeichen in der Abbildung, ausgeschlossen jedoch das V.

Gewöhnlich wird für externe Gleitpunktdaten die folgende Abbildung eingesetzt:

-9.99···99E+99

Die VALUE-Klausel ist für externe Gleitpunktdaten nicht erlaubt.

Übertragungen zu externen Gleitpunktdaten sind gestattet, wenn die Sendefelder zu externen oder internen Dezimaldatenelementen, zu Dualdatenelementen oder zu anderen externen Gleitpunktdatenelementen [1] gehören. Die Übertragungen gehören zur numerischen Kategorie. Umgekehrt sind Übertragungen der Werte externer Gleitpunktdatenelemente zu externen oder internen Dezimaldatenelementen, zu Dualdatenelementen, zu aufbereiteten numerischen Datenelementen und zu anderen externen Gleitpunktdatenelementen [2] erlaubt. Auch diese Übertragungen sind numerischer Natur. Übertragungen, in die alphamerische, alphabetische und aufbereitete alphamerische Datenelemente verwickelt sind, können nicht ausgeführt werden.

Die Anwendung externer Gleitpunktdatenelemente unterliegt in der Regel Restriktionen: Sie werden hauptsächlich für die Ausgabe und die Eingabe von Daten eingesetzt.

17.21. Interne Gleitpunktdaten

Unter einem internen Gleitpunktdatenelement versteht man ein Element, das eine Gleitpunktzahl mit dualer Basis enthält. Für diese Elemente ist die Benutzung der PICTURE-Klausel nicht erlaubt. In der USAGE-Klausel ist COMPUTATIONAL-1 aufzuführen, wenn das Datenelement ein Wort, COMPUTATIONAL-2, wenn es ein Doppelwort belegen soll. Die erweiterte Genauigkeit wird von COBOL nicht unterstützt.

Die Datenübertragungen, die für externe Gleitpunktdatenelemente erlaubt sind (siehe Abschnitt 17.20.) sind auch für interne Gleitpunktdatenelemente gestattet. Hinzu kommen außerdem die Übertragungen zwischen externen und internen Gleitpunktdatenelementen in beiden Richtungen. In diese sind Umwandlungen eingeschlossen, die den Prozeß stark verlangsamen, so daß man sie, wenn irgendwie möglich, tunlichst unterlassen sollte.

In COBOL werden Gleitpunktoperationen gewöhnlich auf diejenigen Fälle beschränkt, für die sie nicht zu umgehen sind. Dazu zählen beispielsweise Operationen mit Gleitpunktdaten, die aus Programmen herrühren, die in PL/1 oder in der FORTRAN-Sprache abgefaßt sind, sowie Prozesse wie das Radizieren (Wurzelziehen), die mit Daten in Festpunktdarstellung nur sehr schwierig zu beherrschen sind.

17.22. Das Statement MOVE

Die allgemeine Form dieses Statements lautet:

$$\underline{\text{MOVE}} \left\{ \begin{matrix} \underline{\text{bez-a}} \\ \underline{\text{lit}} \end{matrix} \right\} \underline{\text{TO}}\ \underline{\text{bez-z}}\ [\underline{\text{bez-z}}] \cdots$$

Unter bez-a und bez-z sind hierbei irgendwelche Bezeichnungen zu verstehen, d.h. Namen irgendeines Datenelementes, das mit einer Stufennummer zwischen 01 und 50 bzw. 77 im Datenteil erklärt ist. Unter lit sind nichtnumerische Literale, numerische Festpunktliterale, numerische Gleitpunktliterale oder figurative Konstanten zu verstehen.

Wenn mehr als eine Zielbezeichnung bez-z niedergeschrieben wird, so unterscheidet sich das Ergebnis, das durch die Ausführung eines solchen Statements erzielt wird,

1) Hinzu kommen Übertragungen, deren Ausgangsfelder zu internen Gleitpunktdatenelementen gehören (siehe nachfolgenden Abschnitt 17.21.).

2) Hinzu kommen Übertragungen zu internen Gleitpunktdatenelementen (siehe nachfolgenden Abschnitt 17.21.).

Ausgangsgruppe der Übertragung		*Ziel der Übertragung*							
	Abkürzung	*Gruppe*	*AM*	*AA*	*AB*	*ED*	*ID*	*DU*	*NA*	*EG*	*IG*
Gruppe	*G*	a	a	a	a					a	
Alphamerisches Datenelement	*AM*	a	a	a	1)					1)	
Nichtnumerisches Literal *Figurative Konstante ALL*		a	a	a[2)]	3)					x	
Alphabetisches Datenelement *Aufbereitetes alphamerisches Datenelement* *Figurative Konstante SPACES*	*AB* *AA*	a	a	a	x					x	
Aufbereitetes numerisches Datenelement *Figurative Konstanten QUOTES HIGH-VALUE LOW-VALUE*	*NA*	a	a	x	x					x	
Figurative Konstante ZEROS		a	a	x	n					n	
Externe Dezimaldatenelemente *Interne Dezimaldatenelemente* *Dualdatenelemente* *Numerische Festpunktliterale*	*ED* *ID* *DU*	a	4)	x	n					n	
Externe Gleitpunktdatenelemente *Interne Gleitpunktdatenelemente* *Numerische Gleitpunktliterale*	*EG* *IG*	a	x	x	n					n	

Bedeutung der Abkürzungen für die Übertragungsarten:

a nichtnumerische (alphamerische) Übertragung
n numerische Übertragung
x Die Übertragung ist nicht erlaubt

Anmerkungen:

1) Das alphamerische Datenelement wird als externes Dezimaldatenelement ohne Vorzeichen und mit ganzen Zahlen als Wertevorrat angesehen; es erfolgt eine numerische Übertragung.
2) Die Konstante bzw. das Literal muß aus alphabetischen Zeichen bestehen; es erfolgt eine nichtnumerische Übertragung.
3) Die Konstante bzw. das Literal darf nur aus Ziffern bestehen und wird als vorzeichenfreier ganzzahliger Wert eines externen Dezimaldatenelementes angesehen; die Übertragung ist numerischer Natur.
4) Das Sendefeld muß eine ganze Zahl enthalten. Sie wird in eine vorzeichenfreie ganze Zahl umgewandelt; danach erfolgt eine nichtnumerische Übertragung. Ist das Empfangsfeld ein alphamerisches Datenelement, bleibt ein Dezimalpunkt im Sendefeld unbeachtet.

Abb. 17.16. Übersicht über die möglichen Übertragungen

nicht von dem Ergebnis der Ausführung mehrerer aufeinanderfolgender individueller MOVE-Statements. Das Statement

```
MOVE ED1 TO ED4 ED3 ED2.
```

ist infolgedessen äquivalent der Statementfolge

```
MOVE ED1 TO ED4.
MOVE ED1 TO ED3.
MOVE ED1 TO ED2.
```

In der Abb. 17.16. werden alle möglichen Kombinationen von Datentypen aufgezeigt, die in Übertragungsoperationen eingehen können. Nicht erlaubte Übertragungen werden bereits von den Kompilierern erkannt, die folgerichtig auch bei der Übersetzung für die Ausgabe einer Nachricht sorgen. In Einzelfällen versuchen die Kompilierer, das Programm zu einem ausführungsfähigen zu bringen, indem sie SPACES oder ZEROS an Stelle inkorrekter figurativer Konstanten einfügen. Meist werden jedoch falsche MOVE-Statements einfach von den Kompilierern eliminiert.

Ist in eine Übertragung eine Gruppe eingeschlossen, so erfolgt stets eine nichtnumerische Übertragung. Die Gruppe selbst wird dabei als ein Element betrachtet, dessen Abbildung nur die Abbildungszeichen X aufweist.

Zusätzlich zu den in Abb. 17.16. vermerkten Einschränkungen des MOVE-Statements ist auch jedes MOVE-Statement ungültig, bei dem sich Sendefeld und Empfangsfeld überlappen. Diese Regel verbietet beispielsweise die Übertragung des Wertes eines Elementes zu einer Gruppe, die das Element selbst enthält, dessen Wert zu übertragen ist.

Aufgabengruppe 46

1) In einem Programm ist das Statement

```
MOVE G TO H.
```

codiert. Hierbei ist G im Datenteil mit

```
PICTURE S9(4)V99
```

erklärt. Gib den Inhalt des Feldes H für jede der nachfolgend aufgeführten Abbildungen von H an!

Werte von G	PICTURE bei H
1234.567	Z(4).9(3)
2332.ØØØ	*(4).9(3)
1.45	Z(4).9(3)
1.45	$(4).9(3)
1.45	*(4).9(3)
1.45	Z(4).99Ø
.Ø26	Z(4).(2)Ø
-.Ø26	Z999.ØØ
-.Ø26	$(4).$$-
-.Ø26	-.PPP99
-.Ø26	-PPP99

2) Es soll das Statement

```
MOVE J TO K.
```

ablaufen. Für K liegt die Erklärung

```
PICTURE X(5)
```

vor. Gib den Inhalt des Feldes K nach Ausführung des MOVE-Statements für jeden Wert von J an! Sollte das Statement für einzelne Fälle ungültig sein, so ist das zu erläutern!

PICTURE von J	Wert von J
X(4)	ACF8
X(7)	A2C4E67
X(3)	QUOTES
-.9999E-99	b.ØØ26E+Ø1
9(4)	ØØ24
S9(4)	-ØØ24
S9(4)V99	-ØØ24.89
S9(7)	ØØØØØØØ
Z(3).99	B12.89
-.9999	-.Ø269

Anmerkung: b bedeutet hier das Leerzeichen

3) Jede der nachfolgend aufgeführten Teile von Datensatzerklärungen ist in eine der nachfolgenden Klassen einzuordnen!

I. Gruppe
II. Alphabetisches Datenelement
III. Alphamerisches Datenelement
IV. Aufbereitetes alphamerisches Datenelement
V. Externes Dezimaldatenelement
VI. Internes Dezimaldatenelement
VII. Externes Gleitpunktdatenelement
VIII. Internes Gleitpunktdatenelement
IX. Dualdatenelement
X. Aufbereitetes numerisches Datenelement

Die Einordnung soll sich erstrecken auf:

```
a) Ø1 A.
b) Ø2 B  COMPUTATIONAL-2.
c) Ø2 C  PICTURE XXBXXX  DISPLAY.
d) Ø2 D  PICTURE A(4).
e) Ø2 e  COMPUTATIONAL-1.
f) Ø2 F  PICTURE +9.99999E+99.
g) Ø2 G.
h) Ø3 H  PICTURE XX9XX.
i) Ø3 I  PICTURE S9(4)  COMPUTATIONAL-3.
j) Ø2 J  PICTURE S9(4)  COMPUTATIONAL.
k) Ø2 K  PICTURE $9(4).
l) Ø2 L  PICTURE +(4).
```

17.23. Arithmetische Statements

Zur Ausführung arithmetischer Operationen können die in Abb. 17.17. aufgeführten Statements herangezogen werden.

In den in Abb. 17.17. zusammengestellten arithmetischen Statements bedeuten

e-1 e-2	Literale oder Bezeichnungen [1]
bez-1 bez-2	Bezeichnungen [1]

Bei allen Operationen nimmt bez-1 das Resultat auf. Bei den Divisionsstatements stellt e-1 den Dividenden (Zähler) und e-2 den Divisor (Nenner) dar.

Folgt bei den arithmetischen Statements für die Addition bzw. Subtraktion mehr als ein Datenelement den Schlüsselwörtern TO bzw. FROM, so wird die Addition bzw. Subtraktion mit allen aufgeführten Elementen durchgeführt. Erscheint hinter den Schlüsselwörtern ADD bzw. SUBTRACT mehr als ein Begriff (Literal oder Bezeichnung), so werden die Werte aller Begriffe zunächst addiert; die so entstandene Summe wird zum Schluß addiert bzw. subtrahiert.

Im Statement COMPUTE ist unter ausdruck irgendein mathematischer Ausdruck zu verstehen, dem eine bestimmte Bedeutung zukommt. Der Ausdruck braucht einfach bloß aus einem Literal oder aus einer Bezeichnung zu bestehen. Klammern können zur Änderung der Reihenfolge der Operationen benutzt werden. Fehlen die Klammern, so gilt entsprechend den arithmetischen Regeln für die Rechenoperationen ein bestimmter Vorrang: Die Operationen werden von links nach rechts angezogen, wobei die Potenzierungen zuerst, die Additionen und Subtraktionen zuletzt durchgeführt werden. Die folgenden fünf verschiedenen Operatoren können verwendet werden:

Operator	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
**	Potenzierung[2]

Die in arithmetischen Statements aufgeführten Bezeichnungen bzw. Literale müssen sich auf numerische Größen beziehen, d.h. auf externe Dezimaldaten, interne Dezimaldaten, Dualdaten, externe Gleitpunktdaten, interne Gleitpunktdaten, numerische Festpunktliterale, numerische Gleitpunktliterale. Die Operanden arithmetischer Operationen dürfen sich niemals überlappen.

Wird ROUNDED nicht angegeben, so wird das Ergebnis nach Erreichung der für dieses spezifizierten Stellenzahl abgeschnitten. Bei Angabe von ROUNDED wird die in der letzten (rechtsbündigen) Ziffernposition des Ergebnisses erscheinende Ziffer um 1 erhöht, wenn die nachfolgende Ziffernposition eine Ziffer zwischen 5 und 9 aufweist.

---

1) Bezeichnungen sind Namen von Datenelementen, die mit einer Stufennummer zwischen 01 und 50 bzw. 77 im Datenteil erklärt sind.

2) Beispiel: $3*|*2 \mathrel{\hat{=}} 3^2 = 9$.

Operation	*Statements*
Addition	ADD {e-1}··· TO {bez-1 [ROUNDED]}··· [ON SIZE ERROR imperativanweisung]
	ADD {e-1} {e-2}··· GIVING bez-1 [ROUNDED] [ON SIZE ERROR imperativanweisung]
Subtraktion	SUBTRACT {e-1}··· FROM {bez-1 [ROUNDED]}··· [ON SIZE ERROR imperativanweisung]
	SUBTRACT {e-1}··· FROM e-2 GIVING bez-1 [ROUNDED] [ON SIZE ERROR imperativanweisung]
Multiplikation	MULTIPLY e-1 BY bez-1 [ROUNDED] [ON SIZE ERROR imperativanweisung]
	MULTIPLY e-1 BY e-2 GIVING bez-1 [ROUNDED] [ON SIZE ERROR imperativanweisung]
Division	DIVIDE e-2 INTO bez-1 [ROUNDED] [ON SIZE ERROR imperativanweisung]
	DIVIDE e-2 INTO e-1 GIVING bez-1 [ROUNDED] [REMAINDER bez-2] [ON SIZE ERROR imperativanweisung]
	DIVIDE e-1 BY e-2 GIVING bez-1 [ROUNDED] [REMAINDER bez-2] [ON SIZE ERROR imperativanweisung]
Alle Operationen	COMPUTE bez-2 [ROUNDED] = ausdruck [ON SIZE ERROR imperativanweisung]

Abb. 17.17. Zusammenfassende Übersicht über die arithmetischen Statements

Wird die Klausel SIZE ERROR weggelassen, so kann das Resultat von Operationen nicht vorausgesagt werden, wenn es wertmäßig zu groß ist, um in die vorgesehene Stellenzahl des Empfangsfeldes hineinzupassen. Ein solches Resultat wird im allgemeinen nicht abgeschnitten; das Empfangsfeld braucht also nicht unbedingt eine gültige Zahl zu enthalten. Versuche, mit einem solchen Wert weiterzuarbeiten, können durchaus zu einer abnormalen Beendigung des Jobs führen. Wird SIZE ERROR niedergeschrieben, so wird das Resultat in einem temporär eingerichteten Zwischengebiet im Hauptspeicher gebildet und dann mit dem absoluten Maximalwert verglichen, den das Ergebnisfeld aufnehmen kann. Ist dieser überschritten, so wird das Ergebnisfeld nicht zerstört und das unter imperativanweisung aufgeführte Statement ausgeführt. Unter einer Imperativanweisung verstehen wir irgendein Statement, dessen Ausführung nicht von einer Bedingung abhängt, wie es z.B. beim IF-Statement der Fall ist. Alle arithmetischen Statements selbst sind grundsätzlich solche Imperativanweisungen, sofern man bei ihnen die Klausel SIZE ERROR wegläßt. Werden sie dagegen mit dieser Klausel notiert, so gelten sie als bedingte Anweisungen. Ein typisches Beispiel (Abb. 17.18.) möge diese Eröterungen abschließen.

Datenteil

```
77  J  PICTURE S999  COMPUTATIONAL-3.
```

Prozedurteil

```
    MOVE 85 TO J.
    :
    ADD 7 TO J  SIZE ERROR MOVE 999 TO J.
    :
```

Abb. 17.18. Beispiel für ein arithmetisches Statement mit der Klausel SIZE ERROR

Durch das ADD-Statement wird auf den jeweils gegenwärtigen Wert von J immer die Zahl 7 addiert, wenn es zur Ausführung dieses Statements kommt, es sei denn, daß der Wert von J größer als 992 ist; in diesem Fall wird J der Wert 999 zugewiesen.

Zu einer Anziehung der Klausel SIZE ERROR kommt es bei Datenelementen mit den USAGE-Klauseln COMPUTATIONAL-1 bzw. COMPUTATIONAL-2 dann und nur dann, wenn bei einer Division der Divisor gleich Null ist.

17.24. Zwischenergebnisse

Zwischenergebnisse werden immer dann gebildet, wenn bei einem Additions- oder Subtraktionsstatement mehr als ein Begriff (Bezeichnung oder Literal) hinter den Schlüsselwörtern ADD bzw. SUBTRACT erscheint, oder wenn ein arithmetischer Ausdruck mehr als zwei Terms (Glieder) enthält, beispielsweise in einem COMPUTE-Statement. Der Kompilierer erzeugt dabei einen Maschinencode, mit dem eine Serie von Operationen durchgeführt wird. Entstehende Resultate werden in solchen Fällen erforderlichenfalls zur Bildung von nachfolgenden Resultaten herangezogen. Die Bedingung SIZE wird während der Zwischenrechnungen niemals aufgesetzt, sie wird ausschließlich nur dann gesetzt, wenn das Endresultat absolut zu groß ist, um in das vorgesehene Empfangsfeld (Resultatfeld) hineinzupassen.

Die Stellenzahl des Feldes, das zur Aufnahme des Zwischenergebnisses auserkoren ist, wird so gewählt,daß es den Wert aufnehmen kann, der sich ergibt, wenn jede in die

Rechnung eingehende Größe ihren stellenmäßig größtmöglichen Wert angenommen hat (bei Divisoren jedoch den stellenmäßig kleinstmöglichen Wert). Das Zwischenergebnisfeld ist umfangmäßig stets das kleinste, das zur Aufnahme des Zwischenergebnisses ausreicht. Die Anzahl der Dezimalstellen wird wie folgt bestimmt:

a) Von den in Additionen bzw. Subtraktionen eingehenden zwei Werten wird diejenige Dezimalstellenzahl genommen, die die größere von den beiden Werten ist.

b) Bei Multiplikationen ergibt sich die Dezimalstellenzahl des Resultats als Summe der Dezimalstellenzahlen der beiden Faktoren.

c) Bei der Division ergibt sich die Dezimalstellenzahl des Ergebnisses wie folgt: Es wird entweder die Differenz der Dezimalstellenzahlen von Dividend (Zähler) und Divisor (Nenner) oder die Dezimalstellenzahl des Empfangsfeldes genommen, je nachdem, welche größer ist.

d) Bei Potenzierungen ergibt sich die Dezimalstellenzahl des Ergebnisses wie folgt: Ist der Exponent ein ganzzahliges positives Literal, so wird der Exponent mit der Anzahl der Dezimalstellen der Basis multipliziert; das Produkt ist gleich der Dezimalstellenzahl des Ergebnisses. In allen anderen Fällen wird das Maximum der Dezimalstellenzahlen des Ergebnisfeldes, der Basis oder des Zwischenresultates als Dezimalstellenzahl für das Ergebnis genommen.

Die Regeln für die Bildung der Dezimalstellen verursachen den Anfängern oft Ärger, besonders bei der Division. In Abb. 17.19. ist hierzu ein Beispiel dargestellt.

Datenteil

```
77  A  PICTURE S999.
77  B  PICTURE S999.
77  C  PICTURE S999.
```

Prozedurteil

```
    COMPUTE  C = A / N * 1ØØ.
```

Abb. 17.19. Beispiel für die Division zur Ermittlung der Dezimalstellen des Quotienten

Der Wert von A sei 3, der von N gleich 8. Für A/N ergibt sich also 0.375. Die Rechnung wird jedoch auf Null Dezimalstellenzahl ausgerichtet; infolgedessen ergibt sich als Zwischenresultat für A/N der Wert 0. Weitergeführt, aus 0.100 folgt 0. Deshalb wird im Endeffekt C der Wert 0 zugewiesen.

Um zu einem brauchbaren Ergebnis zu gelangen, brauchen wir nur für eine andere Reihenfolge der Operationen zu sorgen. Schreiben wir

```
COMPUTE  C = A * 1ØØ / N.
```

so erhalten wir für C den Wert 37. Hätten wir jedoch das Statement in der Form

```
COMPUTE  C ROUNDED = A * 1ØØ / N.
```

geschrieben, so würde das C zugewiesene Ergebnis 38 lauten. Durch Umstellung der Operatoren und Operanden haben wir jedenfalls ein brauchbares Resultat erzielt und damit den untragbaren Wert 0 vermieden.

Die Stellenzahl von Zwischenresultaten ist auf 30 Dezimalziffernstellen beschränkt. Ein Zwischenergebnis, das diese Stellenzahl überschreitet, wird an einem oder an beiden

Enden abgeschnitten. Die Abschneidemethode ist nicht die gleiche bei allen für die Systeme IBM /360 und IBM /370 zur Verfügung stehenden Kompilierern. Auf jeden Fall stellt jedoch das Zwischenresultat eine gültige Zahl dar, mag sie auch wertmäßig nicht korrekt sein. Deshalb wird bei einem solchen Umstand auch die SIZE-Bedingung nicht gesetzt.

Aufgabengruppe 47

1) Im Datenteil eines in COBOL geschriebenen Programmes liegen folgende Erklärungen vor:

```
77 A  PICTURE S9(3)V9(2)  COMPUTATIONAL-3  VALUE -87.32.
77 B  PICTURE S9(5)V9(2)  COMPUTATIONAL-3  VALUE 1Ø87.33.
77 C  PICTURE SV9(4)      COMPUTATIONAL-3  VALUE .ØØ48.
77 D  PICTURE S9(5)V9(2)  COMPUTATIONAL-3  VALUE 2ØØ.
```

Welches Ergebnis wird D durch die Ausführung der folgenden Statements zugewiesen?

a) `ADD A B C TO D.`
b) `ADD A B C TO D ROUNDED.`
c) `SUBTRACT A B C FROM D ROUNDED.`
d) `ADD A B C GIVING D ROUNDED.`
e) `SUBTRACT A C FROM B GIVING D ROUNDED.`
f) `MULTIPLY C BY D.`
g) `MULTIPLY D BY A GIVING D ROUNDED.`
h) `DIVIDE D INTO A GIVING D.`
i) `DIVIDE B BY D GIVING D ROUNDED.`
j) `COMPUTE D = (C * D) ** 2.`

2) Unter Zugrundelegung des Datenteils von Aufg. 1) ist das nach Ausführung der folgenden Statements sich für C bzw. D ergebende Resultat zu nennen!

a) `DIVIDE D INTO B GIVING C ROUNDED REMAINDER D.`
b) `DIVIDE B BY D GIVING C REMAINDER D.`
c) `DIVIDE .ØØ2 INTO A GIVING C ON SIZE ERROR MOVE 1ØØ TO D.`
d) `COMPUTE C = D ** Ø.5 ON SIZE ERROR MOVE 1ØØ TO D.`
e) `COMPUTE C = .25. COMPUTE D = (A + B + C) / C * 1Ø.`
f) `COMPUTE D = (A + B + C) / 2.5. COMPUTE C = .25.`

18. Weitere Sprachelemente von COBOL

18.1. Unterteilungen von COBOL-Programmen

Jedes COBOL-Programm setzt sich aus vier Teilen zusammen. Mit Ausnahme des Erkennungsteils kann jeder andere Teil in Kapitel zerfallen. Kapitel oder, wenn solche nicht vorhanden sind, auch Teile werden in Paragraphen zerlegt. Die Paragraphen selbst setzen sich aus Sätzen [1] zusammen. Ein Satz enthält entweder eine Datenvereinbarung [2] oder er wird aus ein oder mehreren Anweisungen gebildet [3]. Sowohl Datenvereinbarungen als auch Anweisungen bestehen gewöhnlich aus einem Grundteil oder einem erforderlichen Teil sowie aus verschiedenen freigestellt anwendbaren Klauseln.

Die Namen der Teile, der Kapitel und der Paragraphen beginnen stets innerhalb der Randbegrenzung A. Den Teilnamen muß das Schlüsselwort DIVISION und ein Punkt folgen. Den Kapitelnamen schließt sich das Wort SECTION und ein Punkt an. Den Paragraphennamen folgt nur ein Punkt. Die Sätze müssen in der Regel innerhalb der Stellen der Zeilen erscheinen, die bei der Randbegrenzung B beginnen. Eine Ausnahme bilden nur die Datenvereinbarungen: Bei ihnen dürfen die meisten Stufennummern entweder innerhalb der Randbegrenzung A oder innerhalb der Randbegrenzung B stehen; die Eintragung FD und die Stufennummern 01 und 77 müssen jedoch innerhalb der Randbegrenzung A anfangen.

Kapitel- und Paragraphennamen innerhalb des Prozedurteils werden vom Programmierer vergeben, reservierte Wörter darf er jedoch dazu nicht verwenden. In den anderen Programmteilen sind die in der Sprache festgelegten Namen zu benutzen.

Die im Abschnitt 17.5. aufgeführten Interpunktionsregeln wollen wir zunächst ergänzen, damit uns die weiteren Besprechungen leichter fallen.

1) Zwischen aufeinanderfolgende Operanden eines Statements kann ein Komma eingeschoben werden. Derartige Kommas bewirken nur eine Verbesserung der Lesbarkeit eines Programmes.

   Beispiel: `ADD 7, H TO B.`

2) Zwischen den Klauseln eines Statements kann ein Komma oder ein Semikolon stehen. Auch eine solche Zeichensetzung dient nur zur Verbesserung der Lesbarkeit eines Programmes.

   Beispiel: `ADD 7 TO B, ON SIZE ERROR MOVE 8 TO B.`

3) Zwischen Statements innerhalb eines Satzes kann ein Komma oder das reservierte Wort THEN geschrieben werden. Hierdurch wird ebenfalls nur die Lesbarkeit eines Programmes verbessert.

   Beispiel: `ADD 7 TO B THEN MOVE 9 TO C.`

---

1) Zur Unterscheidung von den im Datenteil erscheinenden Datensätzen wollen wir die aus Anweisungen und Vereinbarungen gebildeten Sätze manchmal auch Statementsätze nennen.

2) Auch als Datenerklärung bezeichnet.

3) Anweisungen und Vereinbarungen werden gemeinsam als Statements bezeichnet.

Im Prozedurteil kann sich das Statement GO TO entweder auf einen Kapitelnamen oder auf einen Paragraphennamen beziehen. Ein Kapitel sollte als eine Menge von Paragraphen aufgefaßt werden, die zur Erledigung einer speziellen Aufgabe innerhalb eines Programmes dient.

18.2. Kommentare und Fortsetzungen

Jeder Satz eines COBOL-Programmes kann über beliebig viele Zeilen fortgesetzt werden; auf jeder Zeile muß jedoch mindestens ein Wort erscheinen. Der Begriff Fortsetzungszeile gilt für jede Zeile, über die ein Literal oder ein Satz weitergeführt wird.

Numerische Literale werden von einer Zeile zu einer anderen Zeile fortgesetzt, indem ein Bindestrich in die Spalte 7 der Fortsetzungszeile plaziert wird. Das letzte, vom Leerzeichen unterschiedliche Zeichen in der vorangehenden Zeile wird dabei mit dem ersten, vom Leerzeichen unterschiedlichen Zeichen in der Fortsetzungszeile verknüpft.

Nichtnumerische Literale können über beliebig viele Zeilen fortgesetzt werden, wobei jede Fortsetzungszeile in der Spalte 7 einen Bindestrich aufweisen muß. Ein an beliebiger Stelle nach der Randbegrenzung B stehendes Anführungszeichen signalisiert dabei dem Kompilierer, daß das nächste Zeichen des Literals der darauffolgenden Spalte dieser Zeile zu entnehmen ist. Mit Ausnahme der letzten Zeile erstreckt sich das Literal auf jeder sonstigen Zeile bis zur Spalte 72 einschließlich.

Eine Kommentarzeile wird durch ein Sternzeichen auf Spalte 7 angezeigt. In den Spalten 8 bis 72 einer solchen Zeile können beliebige Informationen enthalten sein. Derartige Zeilen werden als Bemerkungen angesehen. Leerzeilen dürfen irgendwo in einem Programm erscheinen, doch niemals unmittelbar vor einer Fortsetzungszeile.

Innerhalb des Prozedurteiles können Kommentare auch durch Verwendung des NOTE-Statements eingeschoben werden. Es lautet:

```
NOTE kommentar.
```

Jedes auf das reservierte Wort NOTE folgende Zeichen wird als Bestandteil des Kommentars (der Bemerkung) angesehen. Der erste Punkt, dem ein Leerzeichen folgt, zeigt jedoch das Ende des NOTE-Statements an. Erscheint das NOTE-Statement als erstes Statement eines Paragraphen, so wird der gesamte Paragraph als Kommentar betrachtet.

18.3. Spezielle Register

Unter einem speziellen Register versteht man ein Speichergebiet, das vom Kompilierer in ein Programm eingebaut wird; der Programmierer hat darauf keinen Einfluß. Jeder Registername stellt ein reserviertes Wort dar. Mit jedem Register ist eine Abbildung zu dessen Gebrauch verknüpft. Die Namen der speziellen Register werden wie reguläre Datennamen benutzt. Hinsichtlich der Verwendung einiger spezieller Register müssen Begrenzungen beachtet werden.

Zu den speziellen Registern gehören die folgenden:

1) CURRENT-DATE
   Die zugehörige Abbildung heißt X(8). Der in diesem Register gespeicherte Wert setzt sich aus den nachstehenden Bestandteilen zusammen:

   ```
   mm/dd/jj
   ```

Im Tagesdatum bedeuten mm den Monat, dd den Tag und jj das Jahr. CURRENT-DATE kann nur als Sendefeld in einem MOVE-Statement auftreten.

2) TIME-OF-DAY
Zur Tageszeit gehört die Abbildung 9(6). Der in diesem Register gespeicherte Wert setzt sich aus den nachstehenden Bestandteilen zusammen:

hhmmss

Dabei bedeutet hh die Stunde, mm die Minute und ss die Sekunde der laufenden Tageszeit. TIME-OF-DAY kann nur als Sendefeld in einem MOVE-Statement benutzt werden.

3) RETURN-CODE
Zu diesem nur bei Benutzung des Betriebssystems OS einsetzbaren speziellen Register gehört die Abbildung S9(4) sowie die Verwendungsklausel COMPUTATIONAL und der Anfangswert 0. RETURN-CODE kann sowohl als Sendefeld als auch als Empfangsfeld benutzt werden. Bei Ausführung des Statements STOP RUN wird der gegenwärtige Wert an das Betriebssystem OS übergeben. Durch die Statements JOB und EXEC der Jobsteuersprache kann er dann mit Hilfe des Parameters COND abgefragt werden. Es besteht eine stillschweigende Übereinkunft, daß die Rückkehrcodes zu Vielfachen von 4 gewählt werden; der Rückkehrcode 0 sagt aus, daß die Programmausführung erfolgreich war. Der Maximalwert, der als Rückkehrcode dienen kann, ist 4095, der Minimalwert natürlich 0.

4) COM-REG
Zu diesem nur bei Benutzung des Betriebssystemes DOS einsetzbaren speziellen Register gehört die Abbildung X(11). COM-REG kann sowohl als Sendefeld als auch als Empfangsfeld benutzt werden. Es schließt die Bytes 12 bis einschließlich 22 der Verständigungsregion (des Verständigungsbereiches) ein. Bei der Entdeckung eines JOB-Statements der Jobsteuersprache wird dieses Feld auf duale Nullen gesetzt. Es kann zur Übergabe von Informationen von einem Jobstep zu nachfolgenden Jobsteps dienen.

Da die Verwendung bestimmter spezieller Register nicht bei allen Betriebssystemen möglich ist, muß man an dieses Problem denken, wenn es gilt, ein Programm so umzugestalten, daß es auch unter einem anderen Betriebssystem läuft, sei dieses nun ein Betriebssystem der IBM oder nicht.

18.4. Neudefinitionen (Klausel REDEFINES)

Enthält das Dateienkapitel im Datenteil mehr als eine Erklärung mit der Stufennummer 01 für eine Datei, so haben die einzelnen Erklärungen tatsächlich jeweils eine neue Definition des gleichen Satzgebietes zur Folge. Neudefinitionen können dazu verwendet werden, um verschiedene Beschreibungen vorzusehen, beispielsweise einmal die Abbildung 999 und andererseits die Abbildung XXX. Auch können durch sie verschiedene Datengruppen gestaltet werden, so wie es z.B. der Abb. 18.1. zu entnehmen ist.

Da der Kompilierer nicht wissen kann, welche Definition des Gebietes benutzt wurde, wenn ihm Werte zugewiesen wurden, liegt es in der Verantwortung des Programmierers, sich in einem Statement auf die geeignete Definition zu beziehen. Ein anderer Grund für Neudefinitionen liegt darin, daß Datensätze von unterschiedlicher Art vorliegen. In diesem Fall liegt gewöhnlich ein Feld bei allen Sätzen an gleicher Stelle und ist von gleicher Länge; es dient zur Identifizierung der Satzart.

Für den Kompilierer ist der Grund für eine Neudefinition ohne Belang. Er benutzt einfach die passende Länge, den zugehörigen Datentyp usw. für jeden Datennamen.

```
Ø1      A.
  Ø2    B  PICTURE X(8).
  Ø2    C.
    Ø3  D  PICTURE X(4).
    Ø3  E  PICTURE X(6).
  Ø2    F  PICTURE X(2Ø).
Ø1      G.
  Ø2    H  PICTURE X(8).
  Ø2    I.
    Ø3  J  PICTURE X(8).
    Ø3  K  PICTURE X(2).
  Ø2    L  PICTURE X(2Ø).
```

Abb. 18.1. Beispiele für eine Neudefinition im Dateienkapitel des Datenteils

Sind in eine Neudefinition Sätze verwickelt, die im Arbeitsspeicher definiert sind, oder solche, die nicht die Stufennummer 01 aufweisen, so müssen wir die Klausel REDEFINES verwenden. Sie lautet:

stn dn-1 REDEFINES dn-2 ···

Die Klausel REDEFINES muß in der Datenbeschreibung als erste vorliegen. Beide Datennamen dn-1 und dn-2 müssen die gleiche Stufennummer stn besitzen; diese kann zwischen 01 und 49 einschließlich liegen oder gleich 77 sein. Auf eine Ausnahme ist jedoch zu achten, wir haben eigentlich auf diese schon hingewiesen. Die Klausel REDEFINES darf nicht für Daten gebraucht werden, die im Dateienkapitel die Stufennummer 01 besitzen. Für diese liegen bekanntlich die Neudefinitionen implizit vor.

Die Neudefinition beginnt bei dem Element oder der Gruppe mit dem Datennamen dn-1 und setzt sich fort, bis eine Stufennummer gefunden ist, die gleich oder kleiner als die Stufennummer stn ist, die zu dn-1 gehört. Die in Bytes gemessene Länge der neu definierten Daten sollte gleich der Länge der zuerst definierten Daten sein. Die Kompilierer lassen es jedoch zu, daß Neudefinitionen kleinere Längen aufweisen.

In der Abb. 18.2. ist eine Neudefinition gezeigt, die die Schreibarbeit reduziert, vergleicht man sie mit der, die in Abb. 18.1. zu leisten war. Eine Neudefinition kann unmittelbar hinter die Originaldefinition geschrieben werden. In Abb. 18.2. geht beispielsweise die Datengruppe I, eine Neudefinition, dem Datenelement F voraus.

```
Ø1      A.
  Ø2    B  PICTURE X(8).
  Ø2    C.
    Ø3  D  PICTURE X(4).
    Ø3  E  PICTURE X(6).
  Ø2    I  REDEFINES C.
    Ø3  J  PICTURE X(8).
    Ø3  K  PICTURE X(2).
  Ø2    F  PICTURE X(2Ø).
```

Abb. 18.2. Beispiel für eine Neudefinition innerhalb eines Datensatzes (Vergleichsbeispiel zu Abb. 18.1.)

Auf eine Originaldefinition können soviele Neudefinitionen bezogen werden, wie immer man für erforderlich hält. Dabei können die Neudefinitionen in einer beliebigen Reihenfolge niedergeschrieben werden (Abb. 18.3.).

```
Ø2     C.
       :
Ø2     H REDEFINES C.
       :
Ø2     M REDEFINES C.
       :
```

Abb. 18.3. Beispiel für die Verwendung mehrerer Neudefinitionen

Bei Abb. 18.3. darf also auch der neu definierte Datenname H dem neu definierten Datennamen M nachfolgen. Ebenso akzeptieren die Kompilierer auch die Erklärung

```
Ø2   M REDEFINES H.
```

Der Übersichtlichkeit wegen sollte jedoch die Bezugnahme auf C der Bezugnahme auf H vorgezogen werden, d.h. die Verwendung des Originaldatennamens sollte vor der Verwendung eines neu definierten Datennamens rangieren.

Klauseln zur Festlegung von Anfangswerten (VALUE-Klauseln) sind bei einer Neudefinition nicht erlaubt. Ihre Anwendung ist grundsätzlich auf die Originaldefinition eines Gebietes beschränkt.

Neudefinitionen können ineinander geschachtelt sein. Somit kann z.B. in Abb. 18.2. der Definition von K die Eintragung

```
Ø3   N   REDEFINES K   PICTURE 99.
```

folgen. Die Stufennummer 02 bei F schließt dann sowohl die Neudefinition von K als auch die von C ab.

Erfordert das erste Datenelement in einer Neudefinition die Ausrichtung auf eine vom Byte abweichende Hauptspeicherbegrenzung, so muß der Programmierer sichergehen, daß keine Füllbytes vor ihm erforderlich werden. Er kann ja durch FILLER-Angaben für die entsprechende Ausrichtung sorgen.

Das Empfangsfeld in einem Übertragungsstatement oder in einem arithmetischen Statement darf bekanntlich nicht das Sendefeld überlappen. Somit ist, bezugnehmend auf Abb. 18.2., zwar die Niederschrift des Statements

```
MOVE D TO K.
```

gestattet, die des Statements

```
MOVE E TO J.
```

jedoch nicht. Wird das letztere dennoch formuliert, so ist sein Ergebnis nicht voraussagbar.

18.5. Die Definition von Bereichen (OCCURS-Klausel)

Unter einem B e r e i c h versteht man eine Sammlung von Daten, die alle die gleiche Form besitzen, d.h. die gleiche Abbildung und die gleiche Darstellungsform (PICTURE-

und USAGE-Klauseln). Die Bezugnahme auf ein Element des Bereiches erfolgt mit Hilfe von Indizes. Eine Bezugnahme ist deshalb wie folgt niederzuschreiben:

datenname(i-1 [,i-2 [,i-3]])

Ein solcher Datenname heißt auch indizierter Datenname im Gegensatz zu den bisher behandelten nichtindizierten. Unter i-1, i-2, i-3 sind die Indizes [1] zu verstehen; diese können entweder als numerische Literale oder als numerische Datenelemente mit positiven ganzzahligen Werten angegeben werden.

Im Datenteil eines Programmes sind die Bereiche als solche zu kennzeichnen. Der Kompilierer erkennt sie an dem Auftreten der OCCURS-Klausel. Die Grundform dieser Klausel lautet wie folgt:

OCCURS gz TIMES

Hierbei bedeutet gz eine positive ganze (natürliche) Zahl, durch die der Maximalwert festgelegt ist, den der betreffende Index annehmen kann. Die Indizes müssen unbedingt im Wertebereich von 1 bis gz einschließlich liegen.

Die OCCURS-Klausel kann sowohl bei Gruppen als auch bei Datenelementen erscheinen. Datenelemente, die solchen untergeordnet sind, die eine OCCURS-Klausel aufweisen, dürfen ebenfalls eine OCCURS-Klausel besitzen. Derartige Ineinanderschachtelungen (Nestbildungen) sind nur bis zu einem Maximum von drei Klauseln erlaubt, die dann der Reihe nach zum ersten (oder einzigen), zweiten und dritten Index gehören. Nur Datenelemente mit Stufennummern zwischen 02 und 49 einschließlich können durch OCCURS-Klauseln ausgezeichnet sein.

In Abb. 18.4. wird eine Frachttabelle gezeigt, die konstante Werte für drei verschiedene Frachttarife nach vier verschiedenen Entfernungszonen aufweist. Da die VALUE-Klausel nicht für Neudefinitionen erlaubt ist, werden die Anfangswerte zuerst festgelegt. Danach erfolgt erst die Neudefinition unter Einfügung zweier OCCURS-Klauseln, wodurch ein zweidimensionaler Bereich eingerichtet wird.

```
WORKING-STORAGE SECTION.
77    TARIF              PICTURE S9      COMPUTATIONAL.
77    ZONE               PICTURE S9      COMPUTATIONAL.
Ø1    FRACHTTABELLE.
  Ø2  TARIF-A-ZONE-1     PICTURE S9V99   COMPUTATIONAL-3 VALUE +Ø.5Ø.
  Ø2  TARIF-A-ZONE-2     PICTURE S9V99   COMPUTATIONAL-3 VALUE +Ø.5Ø.
  Ø2  TARIF-A-ZONE-3     PICTURE S9V99   COMPUTATIONAL-3 VALUE +Ø.55.
  Ø2  TARIF-A-ZONE-4     PICTURE S9V99   COMPUTATIONAL-3 VALUE +Ø.6Ø.
  Ø2  TARIF-B-ZONE-1     PICTURE S9V99   COMPUTATIONAL-3 VALUE +Ø.6Ø.
  Ø2  TARIF-B-ZONE-2     PICTURE S9V99   COMPUTATIONAL-3 VALUE +Ø.65.
  Ø2  TARIF-B-ZONE-3     PICTURE S9V99   COMPUTATIONAL-3 VALUE +Ø.75.
  Ø2  TARIF-B-ZONE-4     PICTURE S9V99   COMPUTATIONAL-3 VALUE +Ø.85.
  Ø2  TARIF-C-ZONE-1     PICTURE S9V99   COMPUTATIONAL-3 VALUE +Ø.65.
  Ø2  TARIF-C-ZONE-2     PICTURE S9V99   COMPUTATIONAL-2 VALUE +Ø.7Ø.
  Ø2  TARIF-C-ZONE-3     PICTURE S9V99   COMPUTATIONAL-3 VALUE +Ø.8Ø.
  Ø2  TARIF-C-ZONE-4     PICTURE S9V99   COMPUTATIONAL-3 VALUE +Ø.9Ø.
Ø1    FRACHTTABELLE-A  REDEFINES FRACHTTABELLE.
  Ø2    TARIFNUMMER     OCCURS 3.
    Ø3    FRACHT         PICTURE S9V99   COMPUTATIONAL-3 OCCURS 4.
```

Abb. 18.4. Beispiel für die Festlegung eines zweidimensionalen Bereiches mit Anfangswerten: Frachttabelle

1) Bereiche können also ein-, zwei- oder dreidimensional angelegt werden. Eindimensionale Bereiche werden oft als Vektoren, oder Leisten bezeichnet, ein- und zweidimensionale auch als Tabellen oder Tafeln.

Auf die Elemente der Tabelle von Abb. 18.4. können wir uns nur dann beziehen, wenn wir konsequent zwei Indizes niederschreiben. Beispiele hierfür sind

```
FRACHT (TARIF, ZONE)
FRACHT (1, ZONE)
FRACHT (TARIF, 4)
FRACHT (1, 4)
```

Selbstverständlich können wir auch Gruppen ansprechen, beispielsweise die Gruppe, die einen bestimmten Tarif für alle Entfernungszonen beinhaltet. Hierzu nennen wir folgende Beispiele:

```
TARIFNUMMER (TARIF)
TARIFNUMMER (1)
TARIFNUMMER (2)
```

Hinsichtlich der Schreibweise sollten wir uns an einige Interpunktionsregeln erinnern. Mindestens ein Leerzeichen muß vor einer geöffneten Klammer und nach einem Komma stehen; außerdem ist hinter der geschlossenen Klammer ein Leerzeichen oder ein Punkt erforderlich. Außerdem ist zu berücksichtigen, daß einer geöffneten Klammer kein Leerzeichen folgen und einer geschlossenen keines vorangehen darf.

Eintragungen mit der Bezeichnung FILLER werden vielfach für Elemente gebraucht, auf die man sich in einem Programm nicht zu beziehen braucht.

Die Verwendung von Datennamen mit den Angaben COMPUTATIONAL oder COMPUTATIONAL-3 für Indizes führen manchmal zu einem leistungsfähigeren Maschinencode.

Unter Ausnutzung dieser Erkenntnisse wollen wir einen eindimensionalen Bereich definieren; er stellt eine Preistafel dar (Abb. 18.5.).

```
Ø1 PREIS-TAFEL.
    Ø2 FILLER PICTURE 999V99 COMPUTATIONAL-3 VALUE 9.ØØ.
    Ø2 FILLER PICTURE 999V99 COMPUTATIONAL-3 VALUE 12.5Ø.
    Ø2 FILLER PICTURE 999V99 COMPUTATIONAL-3 VALUE 16.ØØ.
    Ø2 FILLER PICTURE 999V99 COMPUTATIONAL-3 VALUE 1Ø.ØØ.
    Ø2 FILLER PICTURE 999V99 COMPUTATIONAL-3 VALUE 14.ØØ.
    Ø2 FILLER PICTURE 999V99 COMPUTATIONAL-3 VALUE 18.ØØ.
Ø1 FILLER REDEFINES PREIS-TAFEL.
    Ø2 PREIS PICTURE 999V99 COMPUTATIONAL-3 OCCURS 6 TIMES.
```

Abb. 18.5. Beispiel für einen eindimensionalen Bereich: Preistafel

Auch beim Beispiel in Abb. 18.5. finden wir erneut die Regel beachtet, daß die Benutzung der REDEFINES-Klausel die Verwendeung der VALUE-Klausel verbietet. Deshalb erscheinen auch die VALUE-Klauseln bei der ersten Definition der Datenelemente, d.h. der Originaldefinition. Der Index könnte im Arbeitsspeicherkapitel beispielsweise mit

```
77  INDEX PICTURE S9 COMPUTATIONAL-4.
```

definiert werden. Wenn wir uns dann auf einen speziellen Preis in der Preistafel beziehen wollen, müssen wir zunächst INDEX einen bestimmten Wert zuweisen, z.B. durch das Statement

```
MOVE 2 TO INDEX.
```

Danach können wir uns auf die zweite Preisangabe durch

```
PREIS (INDEX)
```

beziehen.

18.6. Die Definition von Bereichen mit variablen Dimensionsgrenzen

Bereichen mit einer variablen Anzahl von Elementen werden durch die folgende Form der OCCURS-Klausel festgelegt:

a) OCCURS gz-1 TO gz-2 TIMES
b) OCCURS [gz-1 TO] gz-2 TIMES [DEPENDING ON dn]

Dabei ist zu beachten, daß damit nicht die Länge eines Elementes veränderlich ist, sondern nur die Zahl der Elemente. Sowohl gz-1 als auch gz-2 sind ganze Zahlen, die der Bedingung

$$0 \leq \text{gz-1} < \text{gz-2} \leq 32767$$

genügen müssen. Fehlt gz-1, so wird gz-1 zu 0 angenommen. Durch Benutzung der OCCURS-Klausel in diesen Formen wird ausgesagt, daß der Bereich aus mindestens gz-1 Elementen besteht, jedoch nicht mehr als gz-2 Elemente aufweisen kann. Der dn zugewiesene Wert bestimmt die gerade vorliegende Zahl der Elemente. Ist der Datenname dn zugleich Bestandteil des gleichen Satzes wie der Bereich selbst, so muß dn innerhalb des Datensatzes eher als der erste Bestandteil erscheinen, der selbst eine OCCURS-Klausel mit der Option DEPENDING ON besitzt. Unter dn ist ein nichtnumerisches Element zu verstehen, dessen Wertevorrat nur aus nichtnegativen ganzen Zahlen besteht.

Bei Niederschrift von gz-1 kann die Option DEPENDING ON weggelassen werden, wenn das Programm die Anzahl der Elemente auch ohne Bezugnahme auf dn bestimmen kann. Das ist mitunter der Fall, wenn ein Bereich im Arbeitsspeicher aufgebaut ist und die Anzahl der Elemente aus anderen Datenelementen ermittelt werden kann. DEPENDING ON muß jedoch niedergeschrieben werden, wenn der Umfang des Bereiches vom Steuerprogramm benötigt wird, beispielsweise deshalb, weil der den Bereich enthaltende Datensatz auf ein Ausgabegerät geschrieben wird.

In Abb. 18.6. wird ein Kundenstammsatz vorgestellt, in dem Informationen über nicht erledigte Aufträge enthalten sind; deren Zahl kann zwischen 0 und 23 einschließlich betragen. Ein charakteristisches Verarbeitungsprogramm könnte wie folgt gestaltet sein: Es werden zunächst die alten Kundenstammsätze gelesen, danach werden sie durch Hinzufügung weiterer Aufträge auf den neuesten Stand gebracht, und schließlich werden die aufdatierten Sätze in eine neue Kundenstammdatei geschrieben. Der N zugewiesene Wert würde vom Steuerprogramm dazu benutzt, um die laufende, d.h. gerade aktuelle Satzlänge zu bestimmen. Im Maschinenteil des COBOL-Programmes müßten freilich die Datensätze als zur Satzart V gehörig deklariert sein.

Die Verwendung eines Index, dessen Wert gz-2 überschreitet oder dessen Wert größer ist als der dn zugewiesene Wert, führt zu nicht vorausschaubaren Resultaten, häufig zur anormalen Beendigung des Programmes, d.h. zum Abbruch.

```
WORKING-STORAGE SECTION.
Ø1     KUNDENSTAMMSATZ.
  Ø2   KUNDENBEZEICHNUNG.
    Ø3 KUNDENNUMMER              PICTURE 9(11).
    Ø3 KUNDENNAME                PICTURE X(35).
    Ø3 KUNDENADRESSE-STRASSE     PICTURE X(3Ø).
    Ø3 KUNDENADRESSE-POSTLTZ     PICTURE X(5).
    Ø3 KUNDENADRESSE-WOHNORT     PICTURE X(28).
    Ø3 KUNDENADRESSE-STAAT       PICTURE X(24).
  Ø2   KUNDEN-KONTENSTAND        PICTURE S9(7)V99  COMPUTATIONAL-3.
  Ø2   N                         PICTURE S99       COMPUTATIONAL-3.
  Ø2   AUFTRAG  OCCURS 23        DEPENDING ON N.
    Ø3 AUFTRAGSNUMMER            PICTURE X(6).
    Ø3 AUFTRAGSSCHLUESSEL        PICTURE XX.
    Ø3 MENGE                     PICTURE S9(6)     COMPUTATIONAL-3.
    Ø3 VERSANDDATUM              PICTURE X(1Ø).
```

Abb. 18.6. Beispiel für einen Bereich mit variabler Dimensionsgrenze

18.7. Eintragungen mit besonderen Datenbeschreibungen

In diesem Abschnitt behandeln wir einige Eintragungen, die zur Beschreibung von Datenelementen herangezogen werden können.

Für numerisch aufbereitete Datenelemente (siehe Abschnitt 17.19.) kann die Klausel

BLANK WHEN ZERO

niedergeschrieben werden. Sie hat zur Folge, daß der Kompilierer eine Instruktionsfolge generiert, die dafür sorgt, daß alle Stellen des Datenelementes mit Leerzeichen gefüllt werden, wenn diesem Element die 0 als Wert zugewiesen wird. Bei dieser Ersetzung bleibt die Abbildung unberücksichtigt. Diese Möglichkeit besteht auch bei numerischen Elementen mit der Angabe DISPLAY in der USAGE-Klausel; in diesem besonderen Fall wird ein solches Element als ein numerisch aufbereitetes angesehen.

Alphabetische, alphamerische und alphamerisch aufbereitete Datenelemente (siehe die Abschnitte 17.13. ,17.14. und 17.15.) können durch die Klausel

JUSTIFIED RIGHT

ausgezeichnet werden. Sie bewirkt, daß der Wert, der einem solchen Element zugewiesen wird, rechtsbündig ausgerichtet wird und nicht, wie sonst gewöhnlich, linksbündig. Auffüllen oder Abschneiden erfolgen also im Bedarfsfall am linken Ende des Datenelementes. An Stelle von JUSTIFIED kann abgekürzt auch einfach JUST geschrieben werden.

Für Eintragungen mit der Stufennummer 01 sowie für alle Datenelemente ist es möglich, zusätzlich die Klausel

SYNCHRONIZED [LEFT / RIGHT]

niederzuschreiben. Sie bewirkt, daß Füllbytes vor den Datenelementen eingefügt werden, falls dies notwendig sein sollte, um die Ausrichtung auf die geeignete Hauptspeicherbegrenzung zu erzwingen. Voraussetzung hierfür ist, daß in ihren USAGE-Klauseln nicht DISPLAY vermerkt ist. Bei Angabe von DISPLAY in der USAGE-Klausel wird SYNCHRONIZED ignoriert.

Wird SYNCHRONIZED bei Eintragungen mit der Stufennummer 01 angegeben, so wird es auf jedes Element des Satzes angewendet. Wird diese Klausel dagegen für ein Datenelement niedergeschrieben, so gilt sie ausschließlich nur für dieses allein.

Eine evtl. erfolgte Niederschrift von LEFT oder RIGHT gilt als nicht erfolgt. Als Abkürzung von SYNCHRONIZED kann SYNC benutzt werden.

Fehlt die Klausel SYNCHRONIZED, so erfolgt kein Auffüllen. Bei jeder Bezugnahme auf ein Datenelement, das nicht auf eine geeignete Hauptspeicherbegrenzung ausgerichtet ist, erzeugt dann der Kompilierer zusätzliche Maschineninstruktionen. Sie bewirken, daß der Wert des Datenelementes zunächst in ein Zwischengebiet übertragen wird, wobei er an eine passende Hauptspeicherbegrenzung angeordnet wird. Danach erfolgt seine Verarbeitung. Nach dieser wird er wieder an seinen alten Platz zurückgestellt. Es ist wohl klar, daß diese Instruktionsfolge die Verarbeitungszeit beträchtlich erhöht.

Zusammenfassend sei an dieser Stelle auf bereits bekannte Abkürzungen verwiesen. PIC kann an Stelle von PICTURE benutzt werden. COMP, COMP-1, COMP-2, COMP-3 und COMP-4 können als Abkürzungen von COMPUTATIONAL, COMPUTATIONAL-1 usw. eingesetzt werden.

Die Verwendungsklausel kann auch für Gruppen angegeben werden. In diesem Fall gilt sie für jedes Datenelement in der Gruppe. Wird die USAGE-Klausel zusätzlich auch noch für ein in einer solchen Gruppe enthaltenes Datenelement niedergeschrieben, so muß sie mit der für die Gruppe festgelegten übereinstimmen.

Die VALUE-Klausel kann mit einem nichtnumerischen Literal für eine Gruppe im Arbeitsspeicher vorgesehen werden. Das Literal wird dann als Anfangswert jedem Datenelement der Gruppe zugewiesen. Dabei unterbleibt jegliche Prüfung, ob es sich auch als Wert für das betreffende Element eignet. Die VALUE-Klausel kann in diesem Fall nicht für die Elementarstufe innerhalb der Gruppe spezifiziert werden. An Stelle eines nichtnumerischen Literals kann auch eine figurative (vorgesehene) Konstante benutzt werden.

18.8. Eintragungen mit der Stufennummer 66

Eine beschränkte Form von Neudefinitionen ist mit der Stufennummer 66 vorgesehen. Eine vollständige Eintragung dieser Art lautet wie folgt:

66 d-1 RENAMES d-2 [THRU d-3]

Hierbei bedeuten d-1, d-2 und d-3 nichtindizierte Datennamen. Alle zu einem Datensatz gehörenden Eintragungen mit der Stufennummer 66 müssen unmittelbar auf die letzte Beschreibung eines Datenelementes folgen, das Bestandteil des Satzes ist. Der Datenname d-2 muß eine Stufennummer zwischen 02 und 49 einschließlich besitzen. Wenn d-3 niedergeschrieben wird, muß sich sein Platz innerhalb des Satzes hinter dem von d-2 belegten Platz befinden.

Wenn mit d-2 ein Datenelement angesprochen ist und die THRU-Klausel fehlt, so wird d-1 als Name eines Datenelementes mit der Stufennummer 66 angesehen; damit ist praktisch nur erreicht, daß d-1 ein zweiter Name des mit d-2 benannten Elementes ist.

Verbirgt sich hinter d-2 eine Gruppe und fehlt die THRU-Klausel, so wird d-1 ebenfalls als Gruppe betrachtet, die die gleiche Anzahl von Bytes wie d-1 aufweist.

Bei Angabe der THRU-Klausel wird d-1 als Gruppe angesehen, deren erstes Byte gleich dem ersten Byte von d-2 ist und deren letztes Byte gleich dem letzten Byte von d-3 ist. Ist dabei d-3 selbst eine Gruppe, so gilt als letztes Bytes von d-3 das letzte Byte des letzten Datenelementes von d-3.

In der Abb. 18.7. liegt eine Illustration der RENAMES-Klausel vor. In den meisten Fällen ist es sicherlich genau so einfach die REDEFINES-Klausel an Stelle einer Eintragung mit der Stufennummer 66 zu verwenden.

```
Ø1      A.
  Ø2    B  PICTURE 99.
  Ø2    C.
    Ø3  D  PICTURE X.
    Ø3  E  PICTURE 99.
    Ø3  F  PICTURE XXX.
  Ø2    G  REDEFINES C.
    Ø3  H  PICTURE XX.
    Ø3  J  PICTURE XXXX.
  66    BB  RENAMES B.
  66    CC  RENAMES C.
  66    HE  RENAMES H THRU E.
```

Abb. 18.7. Beispiel für Eintragungen mit der Stufennummer 66

18.9. Eintragungen mit der Stufennummer 88

Eintragungen mit der Stufennummer 88 sind in allen Kapiteln des Datenteils gestattet. Sie weisen die Form

```
88  name  VALUE IS wert
```

auf. Die VALUE-Klausel muß unbedingt niedergeschrieben werden. Unter name wird hierbei nicht ein Datenname, sondern ein Bedingungsname verstanden. Der Bedingungsname kann in einem IF-Statement an Stelle eines regulären Ausdruckes benutzt werden. Um diese Aussagen zu veranschaulichen, wollen wir einmal annehmen, daß im Datenteil die Vereinbarung

```
Ø2  A PICTURE X.
```

vorliegt und im Prozedurteil das Statement

```
IF  A = '3'  THEN  B = 13.
```

erscheint. Eine äquivalente Methode zur Realisierung der gleichen Geschehnisse spiegelt sich in den folgenden Vereinbarungen und Statements wider.

```
Ø2  A  PICTURE X.
88  A3  VALUE '3'.
    IF  A3  THEN  B = 13.
```

Eintragungen mit der Stufennummer 88 müssen unmittelbar im Anschluß an den Datennamen geschrieben werden, auf den sie sich beziehen sollen. Die durch den eingeführten Namen mit der Stufennummer 88 definierte Bedingung ist wahr, wenn das Datenelement, auf das bezuggenommen wird, den spezifizierten Wert angenommen hat. Anderenfalls gilt sie als falsch. In einem IF-Statement ist die Niederschrift eines Bedingungsnamens gleichwertig der Niederschrift

```
datenname = wert-88
```

Unter wert-88 ist hierbei der Wert zu verstehen, der unter wert in der VALUE-Klausel einer Eintragung mit der Stufennummer 88 erscheint.

Eintragungen mit der Stufennummer 88 können sich nicht auf Eintragungen mit der Stufennummer 66 oder andere Eintragungen mit der Stufennummer 88 beziehen. Sie können sich auf Gruppen dann und nur dann beziehen, wenn jedes Element innerhalb der Gruppe in der Verwendungsklausel den Vermerk DISPLAY aufweist und weder die JUSTIFIED- noch die SYNCHRONIZED-Klausel besitzt. Für jeden Datennamen können beliebig viele Eintragungen mit der Stufennummer 88 erfolgen.

Für Eintragungen mit der Stufennummer 88, aber nur für diese, gibt es die VALUE-Klausel in einer erweiterten Form, die wie folgt lautet:

$$\left\{\begin{matrix}\underline{\text{VALUE}}\ \text{IS} \\ \underline{\text{VALUES}}\ \text{ARE}\end{matrix}\right\} \{\underline{\text{literal-1}}\ [\underline{\text{THRU}}\ \underline{\text{literal-2}}]\} \cdots$$

Die Bedingung gilt bei Benutzung dieser Form dann als wahr, wenn der Wert des zugehörigen Datenelementes im Wertebereich von literal-1 bis literal-2, die Grenzen eingeschlossen, liegt. Wird mehr als ein Wertepaar aufgeführt, so gilt die Bedingung dann als wahr, wenn das zugehörige Datenelement einen Wert angenommen hat, der in irgendeinem der genannten Wertebereiche liegt. Fehlt die Angabe von literal-2, so wird unterstellt, daß literal-2 gleich literal-1 ist.

Die Verwendung von Eintragungen mit der Stufennummer 88 befähigt den Programmierer nicht zu neuen Programmiertechniken. Viele Programmierer plagen sich deshalb nicht mit derartigen Eintragungen. Dennoch wäre zu sagen, daß eine gescheite Verwendung von Eintragungen mit der Stufennummer 88 gewisse Arten von Programmänderungen erleichtern kann.

18.10. Gekennzeichnete Namen

In COBOL ist es unter gewissen Voraussetzungen gestattet, daß verschiedene Datenelemente den gleichen Namen aufweisen. Jedes Mal, wenn ein solcher Name in einem Programm gebraucht wird, muß dann dem Kompilierer mitgeteilt werden, auf welches Datenelement er zugreifen soll. Jede Bezugnahme auf einen nicht eindeutigen Datennamen innerhalb eines Programmes muß deshalb gekennzeichnet erfolgen. Die Kennzeichnung erfolgt durch eine Erweiterung des Namens in der Form

$$\left\{\begin{matrix}\underline{\text{IN}} \\ \underline{\text{OF}}\end{matrix}\right\} \underline{\text{datenname-2}}$$

Hierbei bedeutet datenname-2 den Namen einer Gruppe, in der das anzusprechende Datenelement anzutreffen ist. Die beiden Schlüsselwörter IN bzw. OF sind bezüglich ihres Gebrauches als gleichwertig zu betrachten. Bei Eintragungen mit den Stufennummern 01 und 77 scheidet selbstverständlich die Kennzeichnung von vornherein aus, da sie nicht Bestandteil eines umfangreicheren Gebildes sind. Unter datenname-2 kann selbst ein gekennzeichneter Name erscheinen. Somit können wir beispielsweise durchaus ein Statement der Form

```
MOVE  67.3  TO  C OF B IN A.
```

niederschreiben. Es können beliebig viele Kennzeichnungsstufen verwendet werden, sogar dann, wenn die Kennzeichnung zur Identifikation eines Elementes nicht erforderlich ist. Bevor wir mit unseren Erläuterungen fortfahren, wollen wir uns ein simples Beispiel ansehen. Nehmen wir einmal an, daß im Datenteil eines Programmes die in Abb. 18.8. dargestellten Vereinbarungen vorliegen.

```
Ø1  KUNDEN-SATZ.
    Ø2  NAME-ADRESSE           PICTURE X(8Ø).
    Ø2  KONTENSTAND            PICTURE S9(5)V99.
    Ø2  LETZTER-KAUFBETRAG     PICTURE S9(5)V99.
    Ø2  LETZTES-KAUFDATUM      PICTURE X(5).
    Ø2  AUFGELAUFENE-SUMME     PICTURE S9(5)V99.
Ø1  LAUFENDER-KAUF.
    Ø2  NAME-ADRESSE           PICTURE X(8Ø).
    Ø2  LETZTER-KAUFBETRAG     PICTURE S9(5)V99.
    Ø2  LETZTES-KAUFDATUM      PICTURE X(5).
    Ø2  GEZAHLTER-BETRAG       PICTURE S9(5)V99.
```

Abb. 18.8. Auszug aus dem Datenteil eines Programmes

Im Prozedurteil sollen nun die Felder LETZTER-KAUFBETRAG und LETZTES-KAUFDATUM des Stammsatzes KUNDEN-SATZ durch die aktuellen Zahlen aus dem Satz LAUFENDER-KAUF ersetzt werden, damit der Kundenstammsatz wieder auf dem neuesten Stand ist. Außerdem soll der mit KONTENSTAND bezeichnete Saldo des Kundensatzes berichtigt werden, indem der letzte Kaufbetrag addiert und der gezahlte Betrag subtrahiert wird.

Wir finden bei diesem Beispiel, daß zweimal zwei Datenelemente den gleichen Namen besitzen, sie gehören allerdings jeweils zu verschiedenen Sätzen. Deshalb muß dafür gesorgt werden, daß gekennzeichnete Namen benutzt werden, wenn man sich auf sie beziehen will. Unsere Aufgabenstellung kann somit durch die in Abb. 18.9. dargestellte Statementfolge gelöst werden.

```
COMPUTE  KONTENSTAND = KONTENSTAND
                     + LETZTER-KAUFBETRAG IN LAUFENDER-KAUF
                     - GEZAHLTER-BETRAG.
MOVE  LETZTER-KAUFBETRAG IN LAUFENDER-KAUF  TO
      LETZTER-KAUFBETRAG OF KUNDEN-SATZ.
MOVE  LETZTES-KAUFDATUM IN LAUFENDER-KAUF  TO
      LETZTES-KAUFDATUM IN KUNDEN-SATZ.
```

Abb. 18.9. Beispiel für die Benutzung gekennzeichneter Namen in einem Prozedurteil (siehe hierzu auch Abb. 18.8.)

Die Benutzung gekennzeichneter Namen bringt zusätzliche Schreibarbeit mit sich; mitunter kann man sich dadurch aber die Neudefinition von Dateien mit ihren Datensätzen ersparen, die zuvor in anderen Programmen verwendet wurden.

Bei Benutzung von gekennzeichneten Datennamen müssen folgende Regeln beachtet werden:

1) Jede nachfolgende Kennzeichnung muß zu einer höheren hierarchischen Stufe gehören, d.h. sie muß mit einer Stufennummer erklärt sein, die numerisch kleiner ist als die der vorhergehenden Kennzeichnung.
2) Der gleiche Datenname kann nicht auf mehr als einer Stufe innerhalb einer Gruppe auftreten; anderenfalls ist es nämlich unmöglich, eine eindeutige Bezugnahme auf beide Bestandteile der Gruppe herzustellen.
3) Ein mehr als einmal definierter Datenname muß bei seiner Benutzung stets gekennzeichnet werden; eine Ausnahme bildet die REDEFINES-Klausel, durch die die Kennzeichnung bereits eingeschlossen wird.

4) Eine Kennzeichnung kann nicht mit Indizes versehen werden. Der zu kennzeichnende Datenname selbst kann jedoch indiziert werden; in einem solchen Fall sind die Indizes unmittelbar hinter die letzte Kennzeichnung zu schreiben.

5) Kennzeichnungen sind für Datenelemente mit der Stufennummer 88 erlaubt.

6) Ein Name, der keine Kennzeichnung benötigt, darf dennoch gekennzeichnet werden. Kann ein Name auf verschiedene Weisen gekennzeichnet werden, so ist jede Methode zulässig.

In Abb. 18.10. wird eine Definition gezeigt, in der verschiedene Namen mehrmals auftreten. An diese Definition wollen wir einige Betrachtungen knüpfen.

```
Ø1      A.
  Ø2    B  OCCURS 5.
    Ø3  C  PIC S9(3).
Ø1      D.
  Ø2    E.
    Ø3  C  PIC X(7).
    Ø3  F  PIC X(3).
  Ø2    G.
    Ø3  F  PIC 99.
```

Abb. 18.10. Beispiel für eine Definition mit mehreren gleichen Datennamen (Auszug aus dem Datenteil eines Programmes)

Wollen wir uns auf das dritte Element des numerischen Bereiches C, der in der 3. Zeile der Abb. 18.10. aufgeführt ist, beziehen, so können wir uns einer der folgenden Schreibweisen bedienen:

```
C  OF A (3)
C  IN B (3)
```

usw. Eine Bezugnahme auf das alphamerische Datenelement C (Definition durch die Eintragung in der 6. Zeile der Abb. 18.10.) kann mittels der gekennzeichneten Namen

```
C  IN E
C  IN D
C  IN E  IN D
```

usw. erfolgen. Referenzen auf das alphamerische Datenelement F können mittels der gekennzeichneten Namen

```
F  IN E
F  OF E
F  OF E  IN D
```

usw. vorgenommen werden. Zur Bezugnahme auf das numerische Datenelement F bedarf es dagegen der Kennzeichnungen

```
F  IN G
F  OF G
F  IN G  OF D
```

Im Prozedurteil können Paragraphennamen ebenfalls gekennzeichnet sein. Als Kennzeichnung ist hierbei der Kapitelname zu verwenden.

Beispiel:

```
ABC SECTION.
ABD. ...
     :
     :
GO TO ABD IN ABC.
     :
     :
```

Aufgabengruppe 48

1) Ein Bereich ist im Datenteil eines Programmes durch die nachstehenden Vereinbarungen definiert:

```
Ø1      L.
   Ø2   FILLER  PICTURE 99  VALUE ZERO.
   Ø2   FILLER  PICTURE 99  VALUE 77.
   Ø2   FILLER  PICTURE 99  VALUE Ø.
   Ø2   FILLER  PICTURE 99  VALUE 8.
Ø1      FILLER  REDEFINES L.
   Ø2   M  OCCURS 2 TIMES.
     Ø3 N  OCCURS 2 TIMES  PICTURE 99.
Ø1      FILLER  REDEFINES L.
   Ø2   P  OCCURS 2  PICTURE X(4).
```

Gib den Inhalt von L an, wenn der Reihe nach die folgenden Statements ausgeführt werden!

```
a)  MOVE  ZERO  TO  N (1, 1)
b)  MOVE  N (1, 2)  TO  N (2, 2)
c)  MOVE  M (1)  TO  M (2)
d)  MOVE  P (1)  TO P (2)
e)  MOVE  ALL '2'  TO M (2)
f)  MOVE  M (2)  TO  P (1)
g)  MOVE  78  TO  N (2, 2)
h)  MOVE  678  TO  N (2, 2)
i)  MOVE  N (1, 2)  TO  P (2)
j)  MOVE  N (2, 2)  TO  M (1).
```

2) Ein Programm zur Bestimmung des Tilgungsplanes von Hypotheken soll auf der Eingabe von Sätzen fußen, die Hypothekenhöhe, jährlichen Zinsfuß und monatliche Ratenzahlung enthalten. Das Programm soll geeignete Überschriften und den Zahlungsplan ausdrucken. Die Ausgabeliste soll dabei das folgende Aussehen haben:

```
HYPOTHEK:   1ØØØØØ.ØØ DM
ZINSFUSS:       1Ø.5Ø %
MONATSRATE:  1Ø5Ø.ØØ DM

MONAT     RATE      ZINSEN     TILGUNG      KONTENSTAND

  1     1Ø5Ø.ØØ   b875.ØØ     b175.ØØ       1ØØØØØ.ØØ
  2     1Ø5Ø.ØØ      .           .            99825.ØØ
  .        .         .           .               .
  .        .         .           .               .
 1Ø     1Ø5Ø.ØØ
```

Anmerkung: b bedeutet hier das Leerzeichen

3) In einem Programm sollen die Sätze von zwei Dateien gelesen werden.

a) Die Kontenstammdatei ist auf einem Magnetband gespeichert. Ihre Sätze sind wie folgt aufgebaut:

Stellen 1 bis 4: Kontonummer (z.B. BK35)
Stelle 5 : Kennzeichen
Stellen 10 bis 20: Betrag (mit der Abbildung 99999999.99)

Die Sätze sind nach aufsteigenden Kontonummern sortiert. Die Satzlänge ist konstant und beträgt 20 Bytes. Als Kennzeichen dient die Ziffer 1; sie verweist auf den aktuellen (letzten) Kontenstand.

b) Als Bewegungsdatei fungiert ein Lochkartenpaket. Die Karten sind genau so gelocht wie die Stammsätze aufgebaut sind. Es gibt zwei verschiedene Kennzeichnungen auf der Spalte 5:
— 2 bedeutet eine Gutschrift in Höhe des Betrages auf den Spalten 10 bis 20
— 3 bedeutet eine Lastschrift in Höhe des Betrages auf den Spalten 10 bis 20

Die Lochkarten sind ebenfalls nach aufsteigenden Kontonummern sortiert; für jedes Konto können beliebig viele (aber auch keine!) Lochkarten vorliegen. Gutschriften und Lastschriften folgen hingegen in willkürlicher Reihenfolge.

Zur Ausgabe soll eine neue Kontenstammdatei gelangen, eine Stammdatei also, deren Sätze durch Berücksichtigung der Gutschriften und Lastschriften auf den neuesten Stand gebracht sind.

4) Die Aufg. 3) ist wie folgt zu erweitern:
Für jedes Konto, dessen Kontenstand nach der Verarbeitung kein Guthaben aufweist, soll eine Lochkarte gestanzt werden, die genau so aufgebaut ist wie die Lochkarten der Bewegungsdatei. Als Kennzeichen soll auf Spalte 5 die Ziffer 4 gestanzt werden.

5) Die Aufg. 4) ist wie folgt zu erweitern:
Für jedes Konto, dessen Kontenstand nach der Verarbeitung 25000.- DM überschreitet, ist eine Zeile zu drucken, auf der Kontonummer und Kontenstand vermerkt sind.

6) Die Aufg. 4) ist wie folgt zu ergänzen:
Wenn nach der Verarbeitung der Kontenstand eines Kontos gleich 0.00 DM ist, so ist das Betragsfeld der Ausgabekarte frei zu lassen.

18.11. Das Ausrüstungskapitel im Maschinenteil

Der Maschinenteil kann aus zwei Kapiteln bestehen, dem Ausrüstungskapitel und dem Eingabe/Ausgabe-Kapitel, kurz auch EA-Kapitel genannt. Sind beide vorhanden, so geht das Ausrüstungskapitel dem EA-Kapitel voraus. Das Ausrüstungskapitel ist so gestaltet, wie es die Abb. 18.11. zeigt.

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
[SOURCE-COMPUTER  paragraph.]
[OBJECT-COMPUTER  paragraph.]
[SPECIAL-NAMES  paragraph.]
INPUT-OUTPUT SECTION.
```

Ausrüstungskapitel

Abb. 18.11. Das Ausrüstungskapitel im Maschinenteil eines COBOL-Programmes

Der Paragraph, der auf den Quellencomputer verweist, sieht wie folgt aus:

```
SOURCE-COMPUTER. IBM-typ[-modell].
```

Unter typ ist der Typ des Quellencomputers aufzuführen, entweder also 360 oder 370. Für modell ist eine gültige Modellnummer des Computertyps IBM /360 oder IBM /370 zu schreiben. Der gesamte Paragraph wird als Kommentar betrachtet.

Der Paragraph, der auf den Objektcomputer verweist, sieht wie folgt aus:

```
OBJECT-COMPUTER. IBM-36Ø [-modell]
    [MEMORY SIZE IS gz { WORDS      }]
                       { CHARACTERS }
                       { MODULES    }
    [SEGMENT-LIMIT IS anz].
```

Mit Ausnahme der Klausel SEGMENT-LIMIT wird der gesamte übrige Paragraph als Kommentar angesehen. Die in dieser Klausel aufgeführte Größe anz muß aus dem Wertebereich zwischen 0 und 99 einschließlich stammen. Über die Segmentgrenze wollen wir erst im Abschnitt 19.15. diskutieren.

- Die Klausel SEGMENT-LIMIT ist beim Arbeiten mit dem Subset-Kompilierer im DOS nicht einsetzbar.
- Die Kompilierer ab Version 2 gestatten auch die Niederschrift von 370 an Stelle von 360. Dadurch wird veranlaßt, daß der Kompilierer speziell, nur für die IBM /370 geltende Instruktionen generiert, die es also bei der IBM /360 nicht gibt. Damit wird eine schnellere Programmablaufzeit erreicht.

Der Paragraph mit dem Namen SPECIAL-NAMES besitzt die Form:

```
SPECIAL-NAMES.
    [fn IS mn] ...
    [CURRENCY SIGN IS literal]
    [DECIMAL-POINT IS COMMA].
```

Die Niederschrift der letzten Klausel hat zur Folge, daß Punkt und Komma in ihrer regulären Bedeutung bei Abbildungen vertauscht werden. Die Währungsklausel (CURRENCY-Klausel) ermöglicht die Festlegung eines einstelligen nichtnumerischen Literals zur Verwendung in Abbildungen an Stelle des Währungszeichens $. Es dürfen dabei nur Zeichen des Zeichenvorrats von COBOL (siehe Abschnitt 17.3.) angegeben werden.

Unter mn können irgendwelche nichtreservierten Wörter gebracht werden. Diese werden zur Namenszuordnung benutzt, die für bestimmte EA-Statements Gültigkeit besitzt. Jeder mnemonische Name mn ersetzt den eigentlichen Funktionsnamen fn. Es gibt folgende Funktionsnamen:

a) *EA-Einheiten*

CONSOLE	(nur OS)
SYSIN	
SYSIPT	(nur DOS)
SYSPUNCH	
SYSPCH	(nur DOS)
SYSOUT	(nur DOS)
SYSLST	(nur DOS)

b) *Formularsteuerung bei Zeilendruckern*

CSP
CØ1
CØ2
⋮
C12

c) *Ablagefachauswahl bei Lochkartenstanzern*

SØ1
SØ2

Zu den EA-Einheiten wollen wir einige ergänzende Bemerkungen machen. Der Funktionsname CONSOLE verweist auf die normale Systemkonsole, im Betriebssystem DOS als SYSLOG bekannt. Die beiden Funktionsnamen SYSIN bzw. SYSIPT beziehen sich auf die Systemeingabeeinheit (vielfach ein Kartenlesegerät), SYSOUT bzw. SYSLST auf die Systemausgabeeinheit (meist ein Zeilendrucker) und SYSPUNCH bzw. SYSPCH auf den normalen Lochkartenstanzer. Im Betriebssystem OS erscheint der Funktionsname fn in den DD-Steuerstatements der Jobsteuersprache. Im Betriebssystem DOS wird auf die logischen Einheiten SYSLOG, SYSIPT, SYSLST und SYSPCH mit Dateinamen Bezug genommen, wenn sie Plattenspeichergeräten zugeordnet sind (für die letzten drei Einheiten IJSYSIN, IJSYSLS bzw. IJSYSPC).

CSP bewirkt ein Drucken ohne vertikalen Formulartransport, die Funktionsnamen CØ1 bis C12 stehen für Formularvorschübe bis zu einer Lochung in den Kanälen 1 bis 12.

Bei Lochkartenstanzern verweist der Funktionsname SØ2 auf das Ablagefach 2.

Durch entsprechende Eintragungen im Paragraphen SPECIAL-NAMES ist es also möglich, die Funktionsnamen durch leichter merkbare mnemonische Namen nach eigener Wahl zu ersetzen.

18.12. Der Dateisteuerungsparagraph im EA-Kapitel des Maschinenteils

Programme, die auf Eingabe- und/oder Ausgabedateien zugreifen (und das geschieht in nahezu allen Programmen) erfordern die Formulierung des EA-Kapitels im Maschinenteil. Seine Form lautet:

```
INPUT-OUTPUT SECTION.
FILE-CONTROL. asatz ...
[I-O-CONTROL. ssatz ...]
```

In diesem Abschnitt wollen wir den Dateisteuerungsparagraphen (FILE-CONTROL), im nächsten den EA-Steuerungsparagraphen (I-O-CONTROL) behandeln.

Für jede Eingabedatei und für jede Ausgabedatei ist ein Auswahlsatz asatz zu formulieren, der zum Schließen der logischen Verbindung zwischen der COBOL-Datei und der Datenmenge herangezogen wird. Seine Form ist in Abb. 18.12. aufgeführt.

```
SELECT [OPTIONAL] dateiname
ASSIGN TO [gz] systemname-1 [systemname-2] ...
          FOR MULTIPLE { REEL }
                       { UNIT }
[RESERVE { gz } ALTERNATE [AREA ]]
         { NO }           [AREAS]
[ { FILE-LIMIT IS  } {lim-1 THRU lim-2} ... ]
  { FILE-LIMITS ARE}
[ACCESS MODE IS SEQUENTIAL]
[PROCESSING MODE IS SEQUENTIAL]
```

Abb. 18.12. Aufbau der Auswahlsätze (SELECT-Sätze)

In den SELECT-Sätzen können dann einige andere Klauseln erscheinen, wenn die nichtsequentielle Verarbeitung von Dateien vorgesehen ist. Wir werden diese Klauseln zu geeigneter Zeit behandeln.

Im Abschnitt 17.8. haben wir bereits darüber gesprochen, was unter dateiname und systemname, hier systemname-1 und systemname-2, aufzuführen ist. Wird OPTIONAL niedergeschrieben, so wird diese Notiz als Kommentar angesehen. Das gleiche gilt für alle Systemnamen, die zusätzlich zum ersten angegeben sind, d.h. also systemname-2 usw. Auch die Klausel MULTIPLE wird als Kommentar angesehen. Eine unter gz aufgeführte positive ganze Zahl wird im Betriebssystem OS als Kommentar betrachtet; für das Betriebssystem DOS gilt das gleiche, sofern die Datenmengen nicht ohne Kennsätze bzw. mit nicht standardisierten Kennsätzen vorliegen. Ist gz für zur Eingabe dienende kennsatzfreie Datenmengen größer als 1, wird nach Verarbeitung jedes Datenträgers der Operateur zur Montage eines weiteren Datenträgers aufgefordert. Wird gz weggelassen oder ist gz gleich 1, so kann nur eine Magnetbandrolle eingespannt werden. Bei Datenmengen mit nicht standardisierten Kennsätzen wird durch gz die exakte Zahl der zu verarbeitenden Datenträger erfaßt. Im Programm kann dieser Wert überschrieben werden, indem eine davon abweichende Zahl im speziellen Register NSTD-REELS festgelegt wird.

Die Anzahl der EA-Puffer, die für eine Datei eingerichtet werden, ist um 1 größer als die Anzahl der Wechselgebiete, die in der Klausel RESERVE unter gz aufgeführt ist. Im Betriebssystem DOS ist 1 der einzige zulässige Wert für gz; im Betriebssystem OS kann gz als natürliche Zahl zwischen 1 und 254 rangieren. Verzichtet man im OS auf diese Klausel, so ist es möglich, die Zahl der EA-Puffer im DD-Statement der Jobsteuersprache anzugeben. Wird die Zahl der Puffer nirgends festgelegt, so wird ein Wechselgebiet, d.h. zwei Puffer, angenommen.

Die Klausel FILE-LIMIT bzw. FILE-LIMITS wird als Kommentar angesehen. Die beiden Klauseln ACCESS und PROCESSING werden per Standard unterstellt, falls sie fehlen sollten.

Ein typischer Dateisteuerungsparagraph ist in Abb. 18.13. dargestellt.

```
FILE-CONTROL.
    SELECT DATEIA ASSIGN TO UT-S-SYSUT1
    RESERVE NO ALTERNATE AREA.
    SELECT DATEIB ASSIGN TO UR-S-SYSOUT.
```

Abb. 18.13. Beispiel für einen Dateisteuerungsparagraphen

18.13. Der EA-Steuerungsparagraph im EA-Kapitel des Maschinenteils

Der EA-Steuerungsparagraph wird in der Regel weggelassen. Wird er benutzt, so weisen die in ihm enthaltenen Steuerungssätze (im Abschnitt 18.12. als ssatz bezeichnet) den in Abb. 18.14. dargestellten Aufbau auf.

```
SAME AREA FOR dan-1 {dan-2} ···
SAME RECORD AREA FOR dan-1 {dan-2} ···
MULTIPLE FILE TAPE CONTAINS {dan-1 [POSITION gz]} ···
APPLY WRITE-ONLY ON {dan-1} ···
APLLY WRITE-VERIFY ON {dan-1} ···
APPLY RECORD-OVERFLOW ON {dan-1} ···
```

Abb. 18.14. Aufbau der Steuerungssätze im EA-Steuerungsparagraphen

Das Auftreten der Klausel SAME AREA sagt dem Kompilierer, daß zu jedem Zeitpunkt von den aufgeführten Dateien mit den Dateinamen dan-1, dan-2 usw. immer nur eine einzige eröffnet ist. Dadurch wird der Kompilierer in die Lage versetzt, die gleichen EA-Gebiete für alle aufgeführten Dateien bereitzustellen. Keine Datei darf in mehr als einer Klausel SAME AREA erwähnt werden.

Die Klausel SAME RECORD AREA besagt dem Kompilierer, daß sich zu jedem Zeitpunkt nur von einer der aufgeführten Dateien mit den Dateinamen dan-1, dan-2 usw. ein einziger Datensatz im Hauptspeicher befindet. Dadurch kann der Kompilierer ein und dasselbe Satzverarbeitungsgebiet für alle erwähnten Dateien heranziehen. Keine einzige Datei darf mit ihrem Dateinamen in mehr als einer Klausel SAME RECORD AREA erscheinen. Wird eine Datei sowohl in einer Klausel SAME AREA als auch in einer Klausel SAME RECORD AREA benannt, so muß das auch mit allen weiteren Dateien in einer der beiden Klauseln geschehen.

Die Klauseln SAME AREA und SAME RECORD AREA dienen ausschließlich zur ökonomischen Verwaltung, d.h. zur Einsparung von Hauptspeicherplatz.

Wir sollten abschließend darauf hinweisen, daß in der Klausel SAME AREA keine Datei erwähnt werden darf, für die die Klauseln

```
BLOCK CONTAINS Ø
```

oder

```
RECORD CONTAINS Ø
```

vorliegen. Diese beiden Klauseln sind nämlich für die ins Betriebssystem OS eingebauten Kompilierer (Versionen 3 und höher) erlaubt. Im DD-Statement der Jobsteuersprache können dann zur Ausführungszeit die erforderlichen Angaben gemacht werden, wodurch das COBOL-Programm von ihnen befreit ist.

Im Betriebssystem OS wird die Klausel MULTIPLE FILE als Kommentar betrachtet. Das gleiche gilt auch für das Betriebssystem DOS, wenn die angesprochene Datenmenge mit Standardkennsätzen versehen ist. Bei Magnetbändern mit nur einer auf ihnen gespeicherten Datenmenge sollte diese Klausel weggelassen werden. Bei Magnetbändern mit mehreren Datenmengen sind in der Klausel die Namen der Dateien in der richtigen Reihenfolge aufzuführen, d.h. dem Namen der ersten Datei muß der Name der zweiten folgen usw. Werden nicht alle Dateinamen erwähnt, dann wird die unter gz bei POSITION genannte natürliche Zahl als Reihenfolgenummer angesehen. Beim Eröffnen der Datei wird dann das Magnetband um gz minus 1 Bandmarken vorgesetzt.

Die Klausel APPLY WRITE-ONLY gilt nur für Ausgabedateien mit variablen langen Sätzen (Satzformat V). Ihre Niederschrift bewirkt, daß der Ausgabepuffer erst dann zur Ausgabe freigegeben wird, wenn der nächste hinzuzufügende Satz zu lang ist, um noch in den Puffer hineinzupassen. Beim Fehlen dieser Klausel werden die Puffer bereits dann für die Ausgabe bereitgestellt, wenn der verbliebene Platz zu klein ist, um noch einen Satz mit maximaler Satzlänge aufnehmen zu können.

Die Niederschrift der Klausel APPLY WRITE-VERIFY [1] bewirkt im Betriebssystem eine Schreibprüfung bei Ausgabedateien, die auf Plattenspeicher gespeichert werden. Die Schreibprüfung ist standardmäßig für Magnetstreifenspeicher vorgesehen und braucht deshalb für diese nicht angegeben zu werden. Bei Plattenspeichern unterbleibt beim Weglassen dieser Klausel das Zurücklesen der aufgezeichneten Datensätze. Im Betriebssystem OS wird die Schreibprüfung durch eine entsprechende Angabe im DD-Statement der Jobsteuersprache veranlaßt.

Die Klausel APPLY RECORD-OVERFLOW [2] führt im Betriebssystem OS dazu, daß die Sätze auf Plattenspeichern unter Ausnutzung der Spurenüberlaufeinrichtung (Spurenverbindung) geschrieben werden. Somit kann externer Speicherplatz eingespart werden.

---

1) Die Klausel APPLY WRITE-VERIFY steht im Betriebssystem OS nicht zur Verfügung.

2) Die Klausel APPLY RECORD-OVERFLOW steht im Betriebssystem DOS nicht zur Verfügung.

18.14. Eingaben und Anzeigen

Wenige Daten können ohne Definition von Dateien eingegeben werden. Hierzu muß im Prozedurteil das Statement

ACCEPT datenname [FROM { gerätename / mn-name }]

erscheinen. Dieses Statement hat das Lesen eines Satzes zur Folge, dessen Inhalt im Hauptspeicher unter datenname gespeichert wird. Mit datenname kann sowohl eine Gruppe fester Länge als auch ein alphabetisches, alphamerisches, externes dezimales oder externes Gleitpunkt-Datenelement bezeichnet sein. Die Zuweisung des eingelesenen Datensatzes zu datenname erfolgt dabei Byte für Byte von links nach rechts ohne Aufbereitung und ohne Prüfung. Ist der Eingabesatz zu lang, so wird er nach rechts abgeschnitten, ist er zu kurz, so ist das überschüssige rechte Ende von datenname mit nicht voraussehbaren Zeichen aufgefüllt.

Abb. 18.15. zeigt die Gerätenamen, die unter gerätename aufgeführt werden können.

Betriebssystem OS	*Betriebssystem DOS*
CONSOLE SYSIN	a) Kompilierer für die volle Sprache CONSOLE SYSIPT b) Kompilierer für die Subsetsprache CONSOLE

Abb. 18.15. Gerätenamen für das ACCEPT-Statement

Fehlt im ACCEPT-Statement die FROM-Klausel, so wird SYSIN (im OS) bzw. SYSIPT (im DOS) angenommen. Wird unter mn-name ein mnemonischer Name verwendet, so muß dieser im Paragraphen SPECIAL-NAMES definiert sein und sich entweder auf CONSOLE oder aus SYSIN bzw. SYSIPT beziehen.

Bei vorgesehener Konsoleingabe wird zuvor eine Nachricht an das Maschinenbedienungspersonal hinausgeschickt. Die Programmausführung wird solange unterbrochen, bis die Antwort eingelaufen ist. Maximal 114 Zeichen im Betriebssystem OS und 255 im Betriebssystem DOS können eingegeben werden. Bei Eingabe über SYSIN bzw. SYSIPT erfolgt das Einlesen einer 80spaltigen Lochkarte, wobei die Daten auf Spalte 1 beginnen.

Die Ausgabe einer geringen Anzahl von Daten kann durch das Statement

DISPLAY {elem} ... [UPON { gerätename / mn-name }]

veranlaßt werden.

Abb. 18.16. zeigt die Gerätenamen, die im DISPLAY-Statement unter gerätename erscheinen dürfen.

Betriebssystem OS	*Betriebssystem DOS*
CONSOLE SYSPUNCH SYSOUT	a) Kompilierer für die volle Sprache CONSOLE SYSPUNCH SYSPCH SYSLST b) Kompilierer für die Subsetsprache CONSOLE SYSPUNCH SYSPCH

Abb. 18.16. Gerätenamen für das DISPLAY-Statement

Wird die UPON-Klausel weggelassen, so wird SYSOUT (im Betriebssystem OS) bzw. SYSLST (im Betriebssystem DOS) unterstellt. Die Gerätenamen SYSPUNCH und SYSPCH sind äquivalent.

Unter einem Element elem versteht man hier eine figurative Konstante, ein Literal oder ein Datenelement. Ist für ein Element eine Verwendungsklausel (USAGE-Klausel) definiert, in der eine andere Eigenschaft als DISPLAY erklärt ist, so werden die Werte der betreffenden Elemente vor der Ausgabe in einen externen dezimalen Wert oder einen externen Gleitpunktwert umgewandelt. Figurative Konstanten werden als einfaches Erscheinen der Konstanten angesehen. Ist im DISPLAY-Statement mehr als ein Element elem aufgeführt, so werden die Werte der einzelnen Elemente miteinander verkettet. Ist die resultierende Zeichenkette zu lang, um in einen Datensatz hineinzupassen, so werden mehrere Sätze mit jeweils maximaler Satzlänge ausgegeben; der letzte Satz wird nach rechts mit Leerzeichen aufgefüllt, falls das erforderlich ist. Die maximalen Satzlängen betragen:

— 100 Zeichen für CONSOLE
— 120 Zeichen für SYSOUT bzw. SYSLST
— 72 Zeichen für SYSPUNCH bzw. SYSPCH

Auf Zeilendruckern erfolgt die Ausgabe mit einzeiligem Zeilentransport. Bei vorgesehener Lochkartenausgabe wird die Programmidentifikation in die Spalten 73 bis 80 aller Karten gestanzt.

Für mn-name liegt die gleiche Bedeutung vor wie beim ACCEPT-Statement.

Abgesehen von den Versionen 2 und 3 der Kompilierer ist für das ACCEPT-Statement auch die Form

ACCEPT datenname FROM spezielles-register

zulässig; unter spezielles-register können hierbei erscheinen:

a) DATE
Abbildung 9(6): Datum in der Form jjmmdd, wobei jj die letzten zwei Stellen der Jahreszahl, mm den Monat und dd den Tag bedeuten.

b) DAY
Abbildung 9(5): Datum in der Form jjddd, wobei jj die letzten zwei Stellen der Jahreszahl, ddd den laufenden Tag des Jahres bedeuten.

c) TIME
Abbildung 9(8): Zeit in der Form ttmmsshh, wobei tt die Stunden, mm die Minuten, ss die Sekunden und hh die Hundertstelsekunden bedeuten.

18.15. Die Formularsteuerung bei Zeilendruckern

Wenn für die Druckausgabe das Statement

WRITE satzname [FROM bez]

benutzt wird, so erfolgt die Ausgabe mit einzeiligem Zeilentransport. In den meisten Programmen braucht der Programmierer zusätzlichen Einfluß auf die Formularbewegung. Zu diesem Zweck kann er das WRITE-Statement durch eine, aber nur durch eine der beiden folgenden Klauseln ergänzen:

a) { BEFORE / AFTER } ADVANCING { bez-2 LINES / gz LINES / mn-name }

b) AFTER POSITIONING { bez-3 LINES / gz LINES }

Wenn die ADVANCING-Klausel benutzt wird, so muß sie in jedem WRITE-Statement auftreten, das für die betreffende Datei vorgesehen ist. Das gleiche gilt für die POSITIONING-Klausel. Für eine Datei dürfen die beiden Klauseln nicht gemischt hinausgeschickt werden, es gibt nur das entweder-oder. Wird eine der beiden Klauseln verwendet, so muß der Programmierer die Satzlänge um eins vergrößern und als erste Stelle einer Zeile eine von ihm nicht benutzte Stelle zur Aufnahme eines Zeichens vorsehen. Somit muß beispielsweise eine 132stellige Druckzeile mit als 133stelliger Satz erklärt werden, wobei die zu druckenden Zeichen in die rechtsbündigen 132 Stellen hineingestellt werden müssen. Bei Ausführung des WRITE-Statements fügt das Programm ein Steuerzeichen für den Drucker in das erste Byte jeden Satzes ein.

Bei Verwendung der ADVANCING-Klausel muß für gz eine ganze Zahl zwischen 0 und 99 einschließlich angegeben werden. Durch sie wird die Zahl der Zeilen festgelegt, über die das Formular vor oder nach dem Drucken hinweg transportiert werden soll. Während des eigentlichen Druckvorganges erfolgt kein Zeilentransport. Ein Zeilentransport über 0 Zeilen hinweg bedeutet Drucken auf die gleiche Zeile. Wird bez-2 benutzt, so muß es sich bei bez-2 um ein numerisches Datenelement handeln; sein Wertevorrat kann ebenfalls nur zwischen 0 und 99 einschließlich liegen. Bei Verwendung eines mnemonischen Namens unter mn-name muß dieser im Paragraphen SPECIAL-NAMES definiert sein und einem der Funktionsnamen CSP bzw. CØ1 bis C12 zugeordnet werden (siehe hierzu Abschnitt 18.11.). Die Benutzung von mnemonischen Namen ist die einzige Methode, mit deren Hilfe ein Formularvorschub veranlaßt werden kann.

Mnemonische Namen können auch für das Stanzen von Lochkarten verwendet werden. Namen, die SØ1 und SØ2 im Paragraphen SPECIAL-NAMES zugeordnet werden, veranlassen bei ihrem Auftreten in einem WRITE-Statement die Auswahl eines bestimmten Ablagefaches.

In der Klausel POSITIONING darf gz nur gleich 0, 1, 2 oder 3 sein. Diese ganzen Zahlen bewirken folgende Operationen:

0	Vorschub bis zu einer Lochung im Kanal 1 vor dem Drucken
1	Einzeiliger
2	Zweizeiliger } Zeilentransport vor dem Drucken
3	Dreizeiliger

Während des Druckens erfolgt kein Formulartransport, auch kann keiner nach dem Drukken veranlaßt werden. Wird bzw-3 benutzt, so muß diese Bezeichnung zu einem einstelligen alphamerischen Datenelement gehören, das zur Aufnahme der nachfolgenden

ASA-Steuerzeichen [1] für das Drucken bzw. Stanzen vorgesehen ist: b, -, +, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, V, W.

Die zum Drucken vorgesehenen Ausgabedaten werden gewöhnlich zunächst auf Magnetbändern oder Magnetplatten aufgezeichnet und erst später durch ein unabhängiges Programm gedruckt. In denjenigen wenigen Fällen, in denen ein sofortiges direktes Drucken von einem COBOL-Programm veranlaßt werden soll, kann die Klausel

```
AT {END-OF-PAGE} imperativanweisungen
   {EOP        }
```

zu einer der drei Formen des WRITE-Statements hinzugefügt werden. Die Imperativanweisungen (siehe Abschnitt 17.23.) werden dann ausgeführt, wenn die Zeile gedruckt vorliegt, die zu einem Formularvorschub über eine Lochung im Kanal 12 führte. Die Seitenendebedingung tritt dann und nur dann auf, wenn eine Lochung im Kanal 12 durch einen Zeilentransport entdeckt wurde; zu ihrem Auftreten kommt es jedoch nicht, wenn ein Vorschub bis zu einer Lochung im Kanal 12 bzw. darüberhinaus veranlaßt wurde. Die Klausel selbst wird ignoriert, wenn die Ausgabe nicht direkt zu einem Drucker geschickt wird. Die Schlüsselwörter END-OF-PAGE und EOP sind äquivalent.

18.16. Statements für die sequentielle EA

In diesem Abschnitt wollen wir im Detail die EA-Statements besprechen, die für Dateien gelten, die der konsekutiven Organisationsform unterliegen.

Zum Eröffnen von Dateien mit den Dateinamen dan ist das Statement

```
OPEN [INPUT { dan [REVERSED          ] [LEAVE  ] } ...]
            {     [WITH NO REWIND    ] [REREAD ] }
                                       [DISP   ]
     [OUTPUT { dan [WITH NO REWIND ] [LEAVE  ] } ...]
             {                      [REREAD ] }
                                    [DISP   ]
```

einzusetzen. Die beiden Klauseln INPUT und OUTPUT können in beliebiger Reihenfolge niedergeschrieben werden, eine von ihnen muß auf jeden Fall vorhanden sein. Die Leistung eines Objektprogrammes steigt gewöhnlich an, wenn so viele Dateien wie möglich in einem Statement eröffnet werden.

Die Klausel NO REWIND wird als Kommentar angesehen. Die Klausel REVERSED ist nur für Dateien bestimmt, die auf Magnetbändern residieren und rückwärts gelesen werden sollen. Sie kann nicht für Sätze im V- oder U-Format angegeben werden, wenn diese Daten enthalten, für die eine SYNCHRONIZED-Erklärung vorliegt; es sei denn, daß jede Satzlänge durch 8 teilbar ist, werden nicht auf Doppelwortgrenze ausgerichtet.

Die Einstellungsoptionen LEAVE, REREAD und DISP gelten für die Maßnahmen, die bei Datenträgerende (nicht bei Dateiende!) einzuleiten sind. Betroffen hiervon sind nur Magnetbänder. Die Angabe von LEAVE hat zur Folge, daß die Magnetbandrolle auf der Stelle fixiert bleibt, die nach Feststellung des Dateiendes auf dieser Rolle erreicht ist. REREAD bewirkt, daß die Rolle erneut zum Beginn der Datei auf dieser Rolle eingestellt wird. Schließlich verweist DISP darauf, daß die Aktion durchzuführen ist, die in der DISP-Angabe im

---

1) Unter b ist das Leerzeichen zu verstehen.

DD-Statement der Jobsteuersprache festgehalten ist. Für rückwärts zu lesende Dateien besitzen REREAD und LEAVE umgekehrte Bedeutungen.

Im Betriebssystem DOS ist die Angabe der drei Optionen LEAVE, REREAD und DISP nicht gestattet.

Das READ-Statement weist folgende Form auf:

```
READ dan RECORD [INTO bez]
         AT END imperativanweisungen
```

Das READ-Statement kann selbstverständlich nur für Dateien gebraucht werden, für die ein OPEN-Statement mit der INPUT-Klausel vorgelegen hat. Unter dan ist wie üblich der Dateiname zu verstehen; unter bez ist eine Bezeichnung zu verstehen, die mit der Stufennummer 01 erklärt ist. Zu diesem Datengebiet wird der eingelesene Satz übertragen, sobald er im Eingabegebiet verfügbar gemacht wurde. Die Dateiendebedingung AT END tritt beim ersten READ-Statement auf, das auf dasjenige folgt, das den letzten Satz der Datei zur Verarbeitung bereitgestellt hatte. Eingabefehler sind immer unvermeidlich, wenn man sich auf ein Datenelement im Dateienkapitel bezieht, bevor das erste READ-Statement ausgeführt wurde oder nachdem die Dateiendebedingung aufgetreten ist oder nachdem die Datei abgeschlossen wurde. Solche Fehlersituationen ergeben stets nicht vorauszusehende Ergebnisse.

Alle unter imperativanweisungen angegebenen Statements werden nach dem Auftreten der Dateiendebedingung ausgeführt. Das Ende einer solchen Statementgruppe wird durch den Punkt erkannt, der den Satz abschließt.

Das WRITE-Statement besitzt folgende Form:

```
WRITE satzname [FROM bez]
```

Zu dieser Grundform können die bereits bekannten Klauseln (siehe Abschnitt 8.15.) hinzukommen, die für die Steuerung der Zeilendrucker und Lochkartenstanzer gelten. Es kommt zu Fehlern, wenn man sich auf einen Datennamen im Dateienkapitel bezieht, der zu einer nicht eröffneten Datei gehört.

Das CLOSE-Statement dient zum Abschließen der eröffneten Dateien und besitzt folgende allgemeine Form:

```
                                  [WITH NO REWIND   ]
                                  [WITH LOCK        ]
CLOSE { dan [REEL]                [WITH DISP        ] } ...
        [UNIT]                    [WITH POSITIONING ]
```

Nicht alle möglichen Kombinationen von Klauseln sind sinnvoll und daher erlaubt. In der Abb. 18.17. sind die Maßnahmen zusammengestellt, die vom CLOSE-Statement veranlaßt werden. Man beachte hierbei, daß REEL bzw. UNIT das Ende der Datenträgerverarbeitung zur Folge haben. In diesem Fall bleibt die Datei eröffnet und die Verarbeitung fährt mit dem nächsten Datenträger der betreffenden Datei fort. Im Betriebssystem OS werden REEL und UNIT als gleichwertig angesehen; im Betriebssystem DOS ist REEL auf Magnetbänder beschränkt und UNIT auf Datenträger von SEDZ [1].

Die Option LOCK bewirkt über das Abschließen der betreffenden Datei hinaus auch deren Sperre, d.h. sie kann nicht wieder eröffnet werden. LOCK wird ignoriert, wenn es zusammen mit REEL oder UNIT angegeben ist.

---

1) SEDZ ist die Abkürzung von Speichereinheit mit direkter Zugriffsmöglichkeit

Optionen des CLOSE-Statements		Datenträger						
			Letzter Datenträger einer Datei			*Anderer Datenträger einer Datei*		
		Lochkarte, Lochstreifen, Druckzeile etc.	OS	DOS		OS	DOS	
				Band	Platte		Band	Platte
-	-	A	AB	AB	A	AB	AB	A
-	NO REWIND	X	AE	AE	X	AE	AE	X
-	DISP	X	AD	X	X	AD	X	X
-	POSITIONING	X	X	X	X	X	X	X
-	LOCK	AS	ABS	ABS	AS	ABS	ABS	AS
REEL	-	X	X	X	X	UB	UB	X
REEL	NO REWIND	X	X	X	X	UE	UE	X
REEL	DISP	X	X	X	X	X	X	X
REEL	POSITIONING	X	X	X	X	UO	X	X
REEL	LOCK	X	X	X	X	UB	UB	X
UNIT	-	X	X	X	X	UB	X	U
UNIT	NO REWIND	X	X	X	X	UE	X	X
UNIT	DISP	X	X	X	X	X	X	X
UNIT	POSITIONING	X	X	X	X	UO	X	X
UNIT	LOCK	X	X	X	X	UB	X	U

Bedeutung der Zeichen

Zeichen	Bedeutung
A	Abschließen der Datei
U	Abschließen des Datenträgers und Übergang zum nächsten Datenträger
S	Sperren der Datei gegen Wiedereröffnung
B	Einstellung des Datenträgers auf Dateibeginn auf diesem Datenträger
E	Einstellung des Datenträgers auf Dateiende auf diesem Datenträger
O	Einstellung des Datenträgers entsprechend der Einstellungsoption im OPEN-Statement
D	Einstellung des Datenträgers entsprechend der DISP-Angabe im DD-Statement der Jobsteuersprache
X	Ungültige Kombination: Nichtvorausschaubares Resultat

Abb. 18.17. Vom CLOSE-Statement veranlaßte Maßnahmen

18.17. Ein Beispielprogramm

In der Abbildung 18.19. ist ein COBOL-Programm aufgelistet, das zur Rechnungsschreibung benutzt werden soll. Eine durch dieses Programm geschriebene Rechnung ist in Abb. 18.18. dargestellt. Die Kundensätze sind auf einem Magnetband gespeichert und besitzen den gleichen Aufbau wie die Sätze, die von dem in Abb. 17.1. aufgelisteten Programm aufgezeichnet werden. Die Bestellungen sind auf Lochkarten festgehalten, pro Karte eine Bestellung. Die Karten sind wie folgt abgelocht:

Spalten 1 bis 4:	Kundennummer
Spalten 5 bis 16:	Auftragsnummer
Spalten 17 bis 24:	1. Rechnungsposten
Spalten 25 bis 32:	2. Rechnungsposten
Spalten 33 bis 40:	3. Rechnungsposten
Spalten 41 bis 48:	4. Rechnungsposten
Spalten 49 bis 56:	5. Rechnungsposten
Spalten 57 bis 64:	6. Rechnungsposten
Spalten 65 bis 72:	7. Rechnungsposten
Spalten 73 bis 80:	8. Rechnungsposten

Den Bestellkarten geht eine Vorlaufkarte voraus mit dem Datum (Spalten 1 bis 6 in der Form mmttjj [1]) und der Ausgangsrechnungsnummer (Spalten 7 bis 12).

LÄNDLICHE WACHSPRODUKTION GmbH

1553 Grieschnack

Firma

TEUBLER, BRESCHI UND ABST
8013 HAAR BEI MUENCHEN
BEETHOVENSTR. 19-33
BUNDESREP. DEUTSCHLAND

Rechnungsnummer..000011
Datum.....13/05/68
Kundennummer.....0017
Auftrag-Nr.....R 167/05

Menge	Schl.	Verpackungsart	Artikelbezeichnung	Einzelpreis	Gesamtpreis
80	11	KARTON 9X1 KG	ROTGOLD-WACHS	12.50	1,000.00
1,500	23	TROMMEL 210 KG	BODENVERSIEGL.	26.00	39,000.00
1,000	31	KARTON 9X1 KG	GLANZWACHS	20.25	20,250.00
7.314	32	PAKET 10.5 KG	GLANZWACHS	14.00	102,396.00
187,621	12	PAKET 10.5 KG	ROTGOLD-WACHS	13.00	2,439,073.00
7	22	PAKET 10.5 KG	BODENVERSIEGL.	1.65	11.55
		30% RABATT			780,519.16
			Rechnungsbetrag		$1,821,221.39

Abb. 18.18. Beispiel für eine Rechnung, die von dem in Abb. 18.19. aufgelisteten Programm gedruckt wurde

1) Hierbei bedeutet mm Monat, tt Tag und jj die letzten beiden Ziffern des Jahres

Die Felder für die einzelnen Rechnungsposten sind aus den folgenden drei Teilfeldern zusammengesetzt:

a) Die ersten sechs Spalten enthalten die Bestellmenge.
b) Die siebente Spalte enthält den Warenschlüssel.
c) Die letzte Spalte enthält den Verpackungsschlüssel.

Wie man aus dem Datenteil des Programmes ersehen kann, gibt es vier Produkte; jedes ist in drei verschiedenen Verpackungen verfügbar.

Es sind vier Dateien definiert:

a) C Karteneingabedatei
b) T Bandeingabedatei
c) M Bandausgabedatei (ist gleich der auf den neuesten Stand gebrachten Bandeingabedatei)
d) L Druckdatei

Einige Felder des Arbeitsspeichers zeichnen sich durch einen Namen aus, der ihren Zweck eindeutig charakterisiert. Für andere Felder wollen wir jetzt den Verwendungszweck erläutern. Wir benutzen FGC als Programmschalter. Wenn FGC gleich Null ist, dann ist die in den Hauptspeicher eingelesene Karte bereits verarbeitet; es muß deshalb eine neue Karte eingelesen werden. Wenn FGC gleich 1 ist, dann ist die eingelesene Karte noch nicht verarbeitet worden. Bei FGC gleich 2 ist die letzte Karte verarbeitet worden. J wird als Laufvariable benutzt; ihr Wert zeigt auf den Rechnungsposten von den acht möglichen Rechnungsposten einer Karte, der gerade bearbeitet wird. SP wird zur Aufnahme des Steuerzeichens für den Zeilentransport bzw. Vorschub beim Drucker verwendet.

Die Ausgabezeilen für den Drucker werden im Arbeitsspeicher zusammengestellt. Es gibt vier verschiedene Arten von Ausgabezeilen: Überschriftszeilen, Postenzeile, Rabattzeile und die Zeile für den Rechnungsbetrag. Die Rabattzeile wird nur dann gedruckt, wenn ein Rabatt gewährt wird. Man beachte, wie die Tabellen der Einheitspreise und der Bezeichnungen codiert sind.

Im Prozedurteil haben wir alle Paragraphen durch Namen gekennzeichnet, die mit P beginnen. Es sind keine Kapitelnamen vorhanden.

```
ØØ1Ø1Ø IDENTIFICATION DIVISION.
ØØ1Ø2Ø PROGRAM-ID.    CBL2.
ØØ1Ø3Ø AUTHOR.  C. B. G.  ---  AEND. A. G.
ØØ1Ø4Ø DATE-WRITTEN MAERZ 1966.
ØØ1Ø5Ø DATE-COMPILED AUGUST 1966.
ØØ1Ø6Ø*
ØØ1Ø7Ø*
ØØ1Ø8Ø ENVIRONMENT DIVISION.
ØØ1Ø9Ø INPUT-OUTPUT SECTION.
ØØ11ØØ FILE-CONTROL.
ØØ111Ø     SELECT C ASSIGN TO UR-254ØR-S-SYSØ1Ø.
ØØ112Ø     SELECT T ASSIGN TO UT-24ØØ-S-SYSØØ9.
ØØ113Ø     SELECT M ASSIGN TO UT-24ØØ-S-SYSØØ8.
ØØ114Ø     SELECT L ASSIGN TO UR-14Ø3-S-SYSØ11.
ØØ115Ø*
ØØ116Ø*
ØØ117Ø DATA DIVISION.
ØØ118Ø FILE SECTION.
ØØ119Ø FD  C        DATA RECORD IS VORLAUFKARTE,
ØØ12ØØ              LABEL RECORDS ARE OMITTED, RECORDING MODE IS F.
ØØ121Ø Ø1  VORLAUFKARTE.
ØØ122Ø     Ø2  DATUM.
ØØ123Ø         Ø3  MONAT            PICTURE XX.
ØØ124Ø         Ø3  TAG              PICTURE XX.
ØØ125Ø         Ø3  JAHR             PICTURE XX.
ØØ126Ø     Ø2  RECHNUNG             PICTURE 9(6).
ØØ127Ø     Ø2  FILLER               PICTURE X(68).
ØØ128Ø FD  T        DATA RECORD IS EINGABE,
ØØ129Ø              LABEL RECORDS ARE OMITTED, RECORDING MODE IS F.
ØØ13ØØ Ø1  EINGABE.
ØØ131Ø     Ø2  PREIS-KLASSE         PICTURE S9          COMPUTATIONAL.
ØØ132Ø     Ø2  KDR                  PICTURE 9(4).
ØØ133Ø     Ø2  KD-NAME              PICTURE X(35).
ØØ134Ø     Ø2  ADR-ZEILE-1          PICTURE X(35).
ØØ135Ø     Ø2  ADR-ZEILE-2          PICTURE X(35).
ØØ136Ø     Ø2  ADR-ZEILE-3          PICTURE X(35).
ØØ137Ø     Ø2  LETZT-KAUF-DAT       PICTURE X(6).
ØØ138Ø     Ø2  KONTENSTAND          PICTURE S9(7)V99  COMPUTATIONAL-3.
ØØ139Ø     Ø2  FILLER               PICTURE X(3).
ØØ14ØØ FD  M        DATA RECORD IS AUSGABE,
ØØ141Ø              LABEL RECORDS ARE OMITTED, RECORDING MODE IS F.
ØØ142Ø Ø1  AUSGABE                   PICTURE X(16Ø).
ØØ143Ø FD  L        DATA RECORD IS LISTE,
ØØ144Ø              LABEL RECORDS ARE OMITTED, RECORDING MODE IS F.
ØØ145Ø Ø1  LISTE                     PICTURE X(66).
ØØ146Ø*
ØØ147Ø WORKING-STORAGE SECTION.
ØØ148Ø 77  J                        PICTURE S9          COMPUTATIONAL.
ØØ149Ø 77  K                        PICTURE S9          COMPUTATIONAL.
ØØ15ØØ 77  KK                       PICTURE S9          COMPUTATIONAL.
ØØ151Ø 77  FGC                      PICTURE 9    VALUE Ø.
ØØ152Ø 77  RECHN-NR                 PICTURE 9(6).
ØØ153Ø 77  SUMME                    PICTURE S9(7)V99  COMPUTATIONAL-3.
ØØ154Ø 77  RABATT                   PICTURE S9(7)V99  COMPUTATIONAL-3.
ØØ155Ø 77  GESAMTPREIS              PICTURE S9(7)V99  COMPUTATIONAL-3.
ØØ156Ø 77  SP                       PICTURE X.
ØØ157Ø 77  RDATUM                   PICTURE X(6).
ØØ158Ø Ø1  UEBERSCHRIFT.
ØØ159Ø     Ø2  FILLER               PICTURE X.
ØØ16ØØ     Ø2  FILLER               PICTURE X(4)   VALUE SPACES.
```

Abb. 18.19. (Teil 1) Programm zur Rechnungsschreibung in COBOL

```
ØØ2Ø1Ø      Ø2  ADRE                PICTURE X(35).
ØØ2Ø2Ø      Ø2  FILLER              PICTURE X(14)  VALUE SPACES.
ØØ2Ø3Ø      Ø2  ABCDE               PICTURE X(12)  JUSTIFIED RIGHT.
ØØ2Ø4Ø Ø1   EINZ-ZEILE.
ØØ2Ø5Ø      Ø2  FILLER              PICTURE X.
ØØ2Ø6Ø      Ø2  QUANT               PICTURE ZZZ,ZZ9.
ØØ2Ø7Ø      Ø2  FILLER              PICTURE XX     VALUE SPACES.
ØØ2Ø8Ø      Ø2  BESTELL.
ØØ2Ø9Ø          Ø3  BESTELLN        PICTURE 99.
ØØ21ØØ      Ø2  FILLER              PICTURE XX     VALUE SPACES.
ØØ211Ø      Ø2  VERP-BEZ            PICTURE X(15).
ØØ212Ø      Ø2  WARE-BEZ            PICTURE X(15).
ØØ213Ø      Ø2  EINZ-PREIS          PICTURE BBZZZ.99.
ØØ214Ø      Ø2  GES-PREIS           PICTURE BBZ,ZZZ,ZZ9.99.
ØØ215Ø Ø1   RAB-ZEILE.
ØØ216Ø      Ø2  FILLER              PICTURE X(17)  VALUE SPACES.
ØØ217Ø      Ø2  RAB-PROZENT         PICTURE ZZ.
ØØ218Ø      Ø2  FILLER              PICTURE X(1Ø)  VALUE '% RABATT
ØØ219Ø      Ø2  FILLER              PICTURE X(25)  VALUE SPACES.
ØØ22ØØ      Ø2  RAB-BETR            PICTURE Z,ZZZ,ZZZ.ZZ.
ØØ221Ø Ø1   GES-ZEILE.
ØØ222Ø      Ø2  FILLER              PICTURE X(53)  VALUE SPACES.
ØØ223Ø      Ø2  SUMMED              PICTURE SS,SSS,SS9.99.
ØØ224Ø Ø1   VERP-BEZEICHN.
ØØ225Ø      Ø2  FILLER  PICTURE X(15)  VALUE 'KARTON   9X1 KG'.
ØØ226Ø      Ø2  FILLER  PICTURE X(15)  VALUE 'PAKET   1Ø.5 KG'.
ØØ227Ø      Ø2  FILLER  PICTURE X(15)  VALUE 'TROMMEL  21Ø KG'.
ØØ228Ø Ø1   FILLER      REDEFINES VERP-BEZEICHN.
ØØ229Ø      Ø2  VPK     OCCURS 3 TIMES  PICTURE X(15).
ØØ23ØØ Ø1   WAREN-BEZEICHN.
ØØ231Ø      Ø2  FILLER  PICTURE X(15)  VALUE ' ROTGOLD-WACHS '.
ØØ232Ø      Ø2  FILLER  PICTURE X(15)  VALUE ' BODENVERSIEGL.'.
ØØ233Ø      Ø2  FILLER  PICTURE X(15)  VALUE ' GLANZWACHS    '.
ØØ234Ø      Ø2  FILLER  PICTURE X(15)  VALUE ' SUPERFEINWACHS'.
ØØ235Ø Ø1   FILLER      REDEFINES WAREN-BEZEICHN.
ØØ236Ø      Ø2  WRN     OCCURS 4 TIMES  PICTURE X(15).
ØØ237Ø Ø1   PREIS-TABELLE.
ØØ238Ø      Ø2  FILLER  PICTURE 999V99  COMPUTATIONAL-3  VALUE  12.5Ø.
ØØ239Ø      Ø2  FILLER  PICTURE 999V99  COMPUTATIONAL-3  VALUE  13.ØØ.
ØØ24ØØ      Ø2  FILLER  PICTURE 999V99  COMPUTATIONAL-3  VALUE 2ØØ.ØØ.
ØØ241Ø      Ø2  FILLER  PICTURE 999V99  COMPUTATIONAL-3  VALUE   2.3Ø.
ØØ242Ø      Ø2  FILLER  PICTURE 999V99  COMPUTATIONAL-3  VALUE   1.65.
ØØ243Ø      Ø2  FILLER  PICTURE 999V99  COMPUTATIONAL-3  VALUE  26.ØØ.
ØØ244Ø      Ø2  FILLER  PICTURE 999V99  COMPUTATIONAL-3  VALUE  2Ø.25.
ØØ245Ø      Ø2  FILLER  PICTURE 999V99  COMPUTATIONAL-3  VALUE  14.ØØ.
ØØ246Ø      Ø2  FILLER  PICTURE 999V99  COMPUTATIONAL-3  VALUE 18Ø.ØØ.
ØØ247Ø      Ø2  FILLER  PICTURE 999V99  COMPUTATIONAL-3  VALUE  14.ØØ.
ØØ248Ø      Ø2  FILLER  PICTURE 999V99  COMPUTATIONAL-3  VALUE  13.ØØ.
ØØ249Ø      Ø2  FILLER  PICTURE 999V99  COMPUTATIONAL-3  VALUE 21Ø.ØØ.
ØØ25ØØ Ø1   FILLER      REDEFINES PREIS-TABELLE.
ØØ251Ø      Ø2  FILLER  OCCURS 4 TIMES.
ØØ252Ø          Ø3  PREIS  OCCURS 3 TIMES  PICTURE 999V99 COMPUTATIONAL-3.
ØØ253Ø Ø1   RECHN-DATUM.
ØØ254Ø      Ø2  TG                  PICTURE XX.
ØØ255Ø      Ø2  FILLER              PICTURE X      VALUE '/'.
ØØ256Ø      Ø2  MON                 PICTURE XX.
ØØ257Ø      Ø2  FILLER              PICTURE X      VALUE '/'.
ØØ258Ø      Ø2  JHR                 PICTURE XX.
ØØ259Ø Ø1   RAB-PROZ.
ØØ26ØØ      Ø2  FILLER  PICTURE SV99    COMPUTATIONAL-3  VALUE .ØØ.
```

Abb. 18.19. (Teil 2) Programm zur Rechnungsschreibung in COBOL

```
ØØ3ØlØ      Ø2  FILLER  PICTURE SV99    COMPUTATIONAL-3  VALUE .3Ø.
ØØ3Ø2Ø      Ø2  FILLER  PICTURE SV99    COMPUTATIONAL-3  VALUE .Ø8.
ØØ3Ø3Ø      Ø2  FILLER  PICTURE SV99    COMPUTATIONAL-3  VALUE .25.
ØØ3Ø4Ø Ø1   FILLER      REDEFINES RAB-PROZ.
ØØ3Ø5Ø      Ø2  RABP    OCCURS 4 TIMES  PICTURE SV99  COMPUTATIONAL-3.
ØØ3Ø6Ø Ø1   BESTELLUNG.
ØØ3Ø7Ø      Ø2  KUNDE               PICTURE 9(4).
ØØ3Ø8Ø      Ø2  AUFTRAG             PICTURE X(12).
ØØ3Ø9Ø      Ø2  POSTEN  OCCURS 8 TIMES.
ØØ31ØØ          Ø3  MENGE           PICTURE 9(6).
ØØ311Ø          Ø3  MENGEA  REDEFINES MENGE  PICTURE X(6).
ØØ312Ø          Ø3  WARE.
ØØ313Ø              Ø4  ART         PICTURE 9.
ØØ314Ø              Ø4  VERP        PICTURE 9.
ØØ315Ø*
ØØ316Ø*
ØØ317Ø PROCEDURE DIVISION.
ØØ318Ø P1. OPEN  INPUT C, T,  OUTPUT L, M.
ØØ319Ø     READ C   AT END  GO TO P11.
ØØ32ØØ     MOVE DATUM TO RDATUM.
ØØ321Ø     MOVE RECHNUNG TO RECHN-NR.
ØØ322Ø     MOVE MONAT TO MON;  MOVE TAG TO TG;  MOVE JAHR TO JHR.
ØØ323Ø P2. IF  FGC NOT = ZERO  GO TO P3.
ØØ324Ø     MOVE 1 TO FGC.
ØØ325Ø     READ C INTO BESTELLUNG,  AT END  MOVE 2 TO FGC
ØØ326Ø                                      MOVE 9999 TO KUNDE.
ØØ327Ø P3. READ T   AT END  GO TO P13.
ØØ328Ø     GO TO P5.
ØØ329Ø P13.  IF  FGC = 2  GO TO P12.
ØØ33ØØ P4. DISPLAY 'KEINE KUNDEN-NR. BEI ', KUNDE UPON CONSOLE.
ØØ331Ø     READ C INTO BESTELLUNG,  AT END  GO TO P12.
ØØ332Ø     GO TO P4.
ØØ333Ø P5. IF  KDR > KUNDE  GO TO P7.
ØØ334Ø     IF  KDR < KUNDE  GO TO P8.
ØØ335Ø     MOVE ZERO TO FGC.
ØØ336Ø     MOVE KD-NAME TO ADRE;   MOVE RECHN-NR TO ABCDE.
ØØ337Ø     WRITE LISTE FROM UEBERSCHRIFT AFTER POSITIONING Ø.
ØØ338Ø     MOVE ADR-ZEILE-1 TO ADRE;   MOVE RECHN-DATUM TO ABCDE.
ØØ339Ø     WRITE LISTE FROM UEBERSCHRIFT AFTER POSITIONING 1.
ØØ34ØØ     MOVE ADR-ZEILE-2 TO ADRE;   MOVE KUNDE TO ABCDE.
ØØ341Ø     WRITE LISTE FROM UEBERSCHRIFT AFTER POSITIONING 1.
ØØ342Ø     MOVE ADR-ZEILE-3 TO ADRE;   MOVE AUFTRAG TO ABCDE.
ØØ343Ø     WRITE LISTE FROM UEBERSCHRIFT AFTER POSITIONING 1.
ØØ344Ø     ADD 1 TO RECHN-NR.
ØØ345Ø     MOVE RDATUM TO LETZT-KAUF-DAT.
ØØ346Ø     MOVE 8 TO J;   MOVE 2 TO SP;   MOVE ZERO TO SUMME.
ØØ347Ø P6. IF  MENGEA (J) = SPACES  GO TO P9.
ØØ348Ø     MOVE MENGE (J) TO QUANT,   MOVE WARE (J) TO BESTELL.
ØØ349Ø     MOVE ART (J) TO K,   MOVE WRN (K) TO WARE-BEZ.
ØØ35ØØ     MOVE VERP (J) TO KK,   MOVE VPK (KK) TO VERP-BEZ.
ØØ351Ø     MOVE PREIS (K, KK) TO EINZ-PREIS.
ØØ352Ø     MULTIPLY  PREIS (K, KK) BY MENGE (J)  GIVING GESAMTPREIS.
ØØ353Ø     MOVE GESAMTPREIS TO GES-PREIS.
ØØ354Ø     ADD GESAMTPREIS TO SUMME.
ØØ355Ø     WRITE LISTE FROM EINZ-ZEILE AFTER POSITIONING SP.
ØØ356Ø     MOVE SPACE TO SP.
ØØ357Ø     GO TO P9.
ØØ358Ø P7. DISPLAY 'KEINE KUNDEN-NR. BEI ', KUNDE UPON CONSOLE.
ØØ359Ø     READ C INTO BESTELLUNG,  AT END  MOVE 2 TO FGC
ØØ36ØØ                                      MOVE 9999 TO KUNDE.
```

Abb. 18.19. (Teil 3) Programm zur Rechnungsschreibung in COBOL

```
ØØ4Ø1Ø      GO TO P5.
ØØ4Ø2Ø P9.  SUBTRACT 1 FROM J.
ØØ4Ø3Ø      IF  J NOT = ZERO  GO TO P6.
ØØ4Ø4Ø      IF  RABP (PREIS-KLASSE) = ZERO  GO TO P1Ø.
ØØ4Ø5Ø      MULTIPLY  RABP (PREIS-KLASSE) BY 1ØØ  GIVING RAB-PROZENT.
ØØ4Ø6Ø      COMPUTE  RABATT = RABP (PREIS-KLASSE) * SUMME.
ØØ4Ø7Ø      SUBTRACT RABATT FROM SUMME.
ØØ4Ø8Ø      MOVE RABATT TO RAB-BETR.
ØØ4Ø9Ø      WRITE LISTE FROM RAB-ZEILE AFTER POSITIONING 2.
ØØ41ØØ P1Ø.  MOVE 3 TO SP;   MOVE SUMME TO SUMMED.
ØØ411Ø      WRITE LISTE FROM GES-ZEILE AFTER POSITIONING SP.
ØØ412Ø P8.  WRITE AUSGABE FROM EINGABE.
ØØ413Ø      GO TO P2.
ØØ414Ø P11.  DISPLAY 'KEINE VORLAUFKARTE (DATUMSKARTE)' UPON CONSOLE.
ØØ415Ø P12.  CLOSE  C T L M.
ØØ416Ø      STOP RUN.
```

Abb. 18.19. (Teil 4) Programm zur Rechnungsschreibung in COBOL

18.18. Testhilfestatements

Die in diesem Abschnitt zu behandelnden Statements sind geschaffen worden, um dem Programmierer Hilfestellung beim Testen seiner Programme zu leisten.

Während der Ausführung bewirkt das Statement

READY TRACE

das Drucken der vom Kompilierer generierten Zeilennummern von Paragraphen und Kapiteln. Das geschieht in dem Augenblick, in dem die betreffenden Paragraphen bzw. Kapitel betreten werden. Die Ausgabe erfolgt über SYSOUT (Betriebssystem OS) bzw. SYSLST (Betriebssystem DOS). Diese Protokollierung wird während des Programmablaufes solange fortgesetzt, bis ein Statement

RESET TRACE

auftaucht.

Eine formatierte Ausgabe der Werte von Datenelementen wird durch das Statement

$$\text{EXHIBIT} \left\{ \begin{matrix} \text{NAMED} \\ \text{CHANGED NAMED} \\ \text{CHANGED} \end{matrix} \right\} \{\text{elem}\} \cdots$$

erreicht. Beim elem muß es sich entweder um einen Datennamen oder um ein nichtnumerisches Literal handeln. Die Werte von Elementen, die durch Datennamen angesprochen werden, werden genau so ausgegeben wiebeim DISPLAY-Statement. Nichtnumerische Literale werden Zeichen für Zeichen angezeigt; am Schluß folgt ein abschließendes Leerzeichen. Wird NAMED niedergeschrieben, so wird der Name eines jeden Datenelementes zusammen mit seinem Wert gedruckt, anderenfalls werden nur die Werte ausgegeben. Bei Angabe von CHANGED werden die Werte und bei zusätzlicher Angabe von NAMED auch die Namen der Datenelemente generell während der ersten Ausführung des Statements gedruckt; danach erfolgt ihre Ausgabe dann und nur dann, wenn eine Wertänderung gegenüber der letzten Ausführung des Statements eingetreten ist. Fehlt die Niederschrift von CHANGED, so werden die Werte aller aufgeführten Datenelemente stets während der Ausführung dieses Statements gedruckt.

Kein Operand darf mehr als 256 Bytes umfassen. Erscheint ein Datenelement in mehr als einem EXHIBIT-Statement, so behandelt jedes Statement das betreffende Element unabhängig von den anderen Statements.

Das ON-Statement besitzt folgende Form:

```
ON nz-1 [AND EVERY nz-2] [UNTIL nz-3]
    {imperativanweisungen}
    {NEXT SENTENCE       }
  [ {ELSE     } {statementfolge}  ]
  [ {OTHERWISE} {NEXT SENTENCE}   ]
```

Hierbei stellen nz-1, nz-2, nz-3 natürliche Zahlen aus dem Wertebereich zwischen 1 und 16777215 einschließlich dar. [1)]

Die Schlüsselwörter ELSE und OTHERWISE haben die gleiche Bedeutung. Wird die durch sie eingeleitete Klausel weggelassen, so muß das ON-Statement mit einem Punkt abgeschlossen werden; außerdem wird

```
ELSE NEXT SENTENCE
```

angenommen.

Im Programm wird ein Zähler n installiert, der die Ausführungen des ON-Statements zählt. Ist dabei n gleich der Summe aus nz-1 und dem Produkt k mal nz-2, wobei k=0,1,2,··· ist, so wird die Statementfolge imperativanweisungen ausgeführt. Zusätzlich muß die Bedingung n≤nz-3 eingehalten sein. Wird an Stelle von imperativanweisungen die Schlüsselwortfolge NEXT SENTENCE niedergeschrieben, so wird keine Operation veranlaßt. Treffen die Voraussetzungen für die Ausführung der soeben beschriebenen Klausel vor den optionalen Wörtern ELSE bzw. OTHERWISE nicht zu, so wird an ihrer Stelle die Statementfolge ausgeführt, die hinter ELSE bzw. OTHERWISE notiert ist. Unter imperativanweisungen bzw. statementfolge können beliebig viele Statements erscheinen; ihr Abschluß wird durch ELSE oder OTHERWISE bzw. durch den Punkt erkannt, mit dem das Ende des ON-Statements signalisiert wird.

Fehlt die Niederschrift von nz-3, so unterbleibt der Test n≤nz-3. Wird sowohl nz-2 als auch nz-3 vermißt, so gilt die Bedingung nur ein einziges Mal als erfüllt, und zwar für n=nz-1. Und schließlich: Ist nz-3 vorhanden, nz-2 dagegen nicht, so wird nz-2=1 angenommen. Ein Beispiel soll das Verständnis über dieses etwas kompliziertere Statement abrunden.

```
ON  2 AND EVERY 3 UNTIL 9  DISPLAY X
                           ELSE NEXT SENTENCE.
```

Durch dieses ON-Statement wird der Wert von X dann ausgegeben, wenn dieses Statement zum 2., 5. und 8. Mal ausgeführt wird. In allen übrigen Fällen geschieht nichts.

18.19. Die Statements GO TO und ALTER

Das Statement GO TO kann in folgenden drei Formen benutzt werden:

a) GO TO prozedurname
b) GO TO
c) GO TO {prozedurname} ··· DEPENDING ON datenname

---

1) Mit Ausnahme der Kompiliererversionen 2 dürfen nz-1, nz-2 und nz-3 auch numerische Festpunktdatenelemente sein, deren Werte im gleichen Bereich wie die oben genannten natürlichen Zahlen liegen.

Unter prozedurname ist entweder der Name eines Paragraphen oder der Name eines Kapitels zu verstehen. In der Form a) wird die Programmablaufsteuerung an das erste Statement des Programmteils übertragen, auf den durch prozedurname Bezug genommen wird. Die zweite Form kann nicht ohne das ALTER-Statement verwendet werden.

Bei der unter c) genannten Form wird zunächst der Wert des Datenelementes bestimmt, das durch datenname angesprochen wird. Sei dieser gleich n und ist n=1,2,3,···, so wird die Programmablaufsteuerung an den Programmteil übertragen, dessen Name als n-ter hinter GO TO erscheint. Ist n größer als die Anzahl dieser Programmteilnamen oder ist n kleiner als 1, so geht die Programmablaufsteuerung an das Statement über, das auf das Statement GO TO folgt. Zur Erläuterung folgt ein Beispiel:

```
GO TO  P1 P2 P3 P3  DEPENDING X.
```

Besitzt X den Wert 1, so erfolgt ein Sprung nach dem mit P1 bezeichneten Paragraphen oder Kapitel. Bei X gleich 2, wird P2 angesprungen, bei X gleich 3 oder X gleich 4, ist dagegen P3 an der Reihe. Bei allen anderen Werten von X wird der nächste Satz des Programmes ausgeführt, d.h. der auf GO TO folgende Satz.

Das Statement ALTER besitzt folgende Form:

```
ALTER  paragraphenname TO [PROCEED TO] prozedurname  ···
```

Unter paragraphenname ist der Name eines Paragraphen aufzuführen, der nur aus einem einzigen Satz besteht, und zwar aus einem Statement GO TO der oben aufgeführten Form a) oder der Form b). Als prozedurname kann der Name irgendeines Paragraphen oder irgendeines Kapitels angegeben werden. Die Wirkung des ALTER-Statements besteht darin, daß der Programmteilname, der unter prozedurname im Statement GO TO auftritt, durch den Namen ersetzt wird, der im ALTER-Statement unter prozedurname steht. Zur Erläuterung möge die in Abb. 18.20. dargestellte Statementfolge beitragen.

```
      .
      .
      ALTER  A TO D,  B TO D.
      .
      .
A.    GO TO.
      .
      .
B.    GO TO E.
      .
      .
```

Abb. 18.20. Beispiel für die Wechselwirkung zwischen den Statements GO TO und ALTER

Werfen wir einen Blick auf die Abb. 18.20. Nach Ausführung des ALTER-Statements werden die beiden Statements GO TO so aufgefaßt, als ob sie in der Form

```
A.  GO TO D.
B.  GO TO D.
```

niedergeschrieben worden wären.

Ein nach der Form b) codiertes Statement GO TO kann nicht vor dem Zeitpunkt ausgeführt werden, zu dem ein ALTER-Statement abgelaufen ist, das auf dieses Statement Bezug genommen hat. Nur durch das ALTER-Statement wird nämlich einem solchen unbedingten Sprung das Sprungziel mitgeteilt.

18.20. Die Beendigung von Programmen

Der normale Abschluß von in COBOL geschriebenen Programmen wird durch die Niederschrift des Statements

STOP RUN

oder durch die Niederschrift des Statements

GOBACK

erreicht. In jedem Fall wird die Programmablaufsteuerung zurück an das Betriebssystem gegeben. Diese beiden Statements können ausgetauscht werden.

Die Ausführung des Statements

STOP literal

hat zur Folge, daß das genannte Literal auf der Bedienerkonsole angezeigt wird. Der weitere Programmablauf wird anschließend aufgeschoben, bis der Maschinenbediener eine Antwort eingegeben hat. Danach wird das Programm weiter ausgeführt. Die Bedienerantwort wird nicht an das COBOL-Programm weitergereicht. Unter literal darf irgendein numerisches oder nichtnumerisches Literal angegeben werden; figurative Konstanten mit Ausnahme von ALL sind ebenfalls zulässig. Die Verwendung dieses Statements beeinträchtigt natürlich die Leistungsfähigkeit des Objektprogrammes.

Aufgabengruppe 49

1) Durch das in Abb. 17.1. dargestellte Programm wird eine Datenmenge auf einem Magnetband erzeugt; das Programm, das in Abb. 18.19. gezeigt ist, druckt einerseits Rechnungen und bringt andererseits die Magnetbanddatenmenge auf den neuesten Stand.

   Es ist nunmehr ein Programm zu schreiben, das die Datenmenge auf dem Magnetband verwaltet, indem es a) Hinzufügungen von Kunden zuläßt und b) eingegangene Zahlungen zum Ausgleich von Krediten auf die Kundensätze verbucht. Zur Eingabe gelangen Lochkarten, die genau so abgelocht sind wie die für das Programm von Abb. 17.1. vorgesehenen. Bei Hinzufügungen neuer Konten enthält die Spalte 5 der ersten Karte die Ziffern 1, 2, 3 oder 4, die Spalte 5 der zweiten Karte die Ziffer 8. Bei einer eingegangenen Zahlung ist die Spalte 5 mit einer 9 versehen, und der Betrag ist auf den Spalten 6 bis 15 untergebracht, wobei die Abbildung 9(7)V99 zugrundegelegt ist. Die Eingabekarten sind aufsteigend nach den Spalten 1 bis 5 sortiert, wobei auf den Spalten 1 bis 4 aller Karten die Kundennummer enthalten ist.

   Das zu schreibende Programm soll die Sätze der alten Datenmenge vom Band und die Lochkarten lesen und eine neue aufdatierte Datenmenge wiederum auf ein Magnetband schreiben.

2) Das durch die Lösung der Aufg. 1) erzeugte Magnetband soll als Eingabe für ein Programm dienen, das monatlich eine Kundenübersicht produziert. Dabei wird für jeden Kunden ein Kontoauszug erstellt, in dem das Berichtsdatum (in der Form 13/Ø5/73) sowie sein Name, seine Adresse, seine Kundennummer und sein Kontostand verzeichnet sind. Hinter dem Kontenstand soll CR gedruckt werden, wenn der Kunde ein Guthaben besitzt. Für Kunden, deren Kontenstand gleich Null ist, soll das Drucken des Kontoauszuges unterbleiben. Das auf den Kontoauszügen angegebene Berichtsdatum soll über eine Vorlaufkarte eingelesen werden.

3) Die Aufgabe 1) ist wie folgt zu modifizieren: Es soll die Eingabe von Karten zugelassen werden, die auf Spalte 5 die Ziffer 7 enthalten. Durch sie wird angezeigt, daß das Kundenkonto gelöscht werden soll. Immer, wenn es zu einer solchen Löschung kommt, ist der Inhalt des auf dem Magnetband gespeicherten Satzes in irgendeiner zweckmäßig aufbereiteten Form zu drucken, damit bei späteren Rückfragen eine Auskunft über das erloschene Konto erteilt werden kann. Das Datum der Löschung soll ebenfalls gedruckt werden.

4) Die Aufg. 3) ist wie folgt zu ergänzen: Am Schluß der Programmausführung ist die Gesamtsumme aller offenen Rechnungen und aller bezahlten Rechnungen zu drucken, sowie ferner die Zahl der Kunden, die zu den beiden Kategorien gehören. Außerdem sind folgende Zahlen zu ermitteln und auszugeben:

   a) Gesamtzahl der auf dem Magnetband gespeicherten Rechnungen
   b) Anzahl der hinzugefügten neuen Kundensätze
   c) Anzahl der gelöschten Kundensätze.

18.21. Das IF-Statement

Das am häufigsten benötigte bedingte Statement ist das IF-Statement, dessen Form allgemein wie folgt lautet:

```
IF  bedingung  THEN  { statementf-1 }
                     { NEXT SENTENCE }

                     { ELSE      }  { statementf-2 }
                     { OTHERWISE }  { NEXT SENTENCE }
```

Es folgt als Beispiel ein typisches IF-Statement:

```
IF  MENGE = ZERO  THEN  COMPUTE PREIS = ZERO
                  ELSE  COMPUTE PREIS = MENGE * EINZ-PREIS
```

Die reservierten Wörter ELSE und OTHERWISE haben die gleiche Bedeutung und können daher auch wechselweise gebraucht werden. Fehlt die ELSE-Klausel, so wird

```
ELSE  NEXT SENTENCE
```

standardmäßig unterstellt. Sowohl statementf-1 als auch statementf-2 stellen Statementfolgen dar; sie können also aus einem oder aus mehreren Statements bestehen; jedes von ihnen muß entweder eine Imperativanweisung (siehe hierzu Abschnitt 17.23.) oder eine weitere IF-Anweisung [1] sein. Die Beendigung von statementf-1 wird durch das Auftauchen von ELSE oder OTHERWISE bzw. bei Fehlen dieser Klausel durch das Vorhandensein eines Punktes erkannt. Die Beendigung von statementf-2 ist durch einen Punkt zu markieren. Die Wortfolge NEXT SENTENCE bedeutet, daß die Programmablaufsteuerung zum nächsten Satz übergeben werden soll, keine andere Maßnahme soll stattfinden.

Während der Ausführung wird zunächst bedingung ausgewertet. Ist sie wahr, so wird die Statementfolge statementf-1 ausgeführt und die Folge statementf-2 übergangen; ist sie falsch, so wird statementf-1 übergangen und statementf-2 (sofern vorhanden) ausgeführt. Wenn in einem Zweig keine Statements ausgeführt werden sollen, so kann also in diesem Zweig NEXT SENTENCE geschrieben werden.

---

1) Für Statement verwenden wir mitunter auch den Begriff Anweisung.

Enthalten statementf-1 bzw. statementf-2 selbst neue IF-Statements, so spricht man von einer Verschachtelung von IF-Statements, auch IF-Nest genannt. In einem solchen Fall wird angenommen, daß jede ELSE-Klausel zum letzten vorangegangenen IF gehört, wenn man das Statement von links nach rechts liest. Werden dabei weniger ELSE-Klauseln als Schlüsselwörter IF festgestellt, so wird

```
ELSE NEXT SENTENCE
```

für alle verbleibenden IF-Statements angenommen. Diese Regel soll nun an einem Beispiel demonstriert werden (Abb. 18.21.).

```
IF  A  THEN  IF  B  THEN  IF  C  THEN  ADD 7 TO X
                                 ELSE  ADD 6 TO X
                          ELSE   IF  C  THEN  ADD 5 TO X
                                 ELSE  ADD 4 TO X.
```

Abb. 18.21. Beispiel für ein IF-Nest

Die durch Ausführung dieses Statementsatzes auf das Datenelement X addierten Werte sind aus der Abb. 18.22. zu entnehmen.

Auswertungsergebnisse der Bedingungen			Auf X addierter Wert
A	B	C	
wahr	*wahr*	*wahr*	7
wahr	*wahr*	*falsch*	6
wahr	*falsch*	*wahr*	5
wahr	*falsch*	*falsch*	4
falsch	*wahr* oder *falsch*	*wahr* oder *falsch*	keine Addition

Abb. 18.22. Ergebnisse der Ausführung des Statementsatzes von Abb. 18.21.

18.22. Vergleichsbedingungen

Eine Vergleichsbedingung besitzt die Form:

```
ausdruck-1  IS [NOT] operator  ausdruck-2
```

Hierbei bedeuten ausdruck-1 und ausdruck-2 irgendwelche arithmetischen Ausdrücke, Literale, figurative Konstanten oder einfache Datennamen. Als Operator können unter operator die folgenden notiert werden.

```
>
<
=
GREATER THAN
LESS THAN
EQUAL TO
```

Zwei verschiedene Vergleichsarten können dabei initialisiert werden. Bei einem numerischen Vergleich (Vergleich numerischer Elemente) wird der Vergleich auf algebraischem Weg durchgeführt. Bei einem nichtnumerischen Vergleich erfolgt der Vergleich auf der Basis der binären Sortierfolge des EBCDI-Codes. Nichtnumerische Vergleiche laufen von links nach rechts ab, wobei der kürzere Operand nach rechts mit Leerzeichen aufgefüllt wird. In der Abb. 18.23. sind die möglichen Vergleiche zusammengefaßt. Genau so wie beim MOVE-Statement können dabei nicht alle Datentypen miteinander kombiniert werden.

1. Vergleichsoperand		*2. Vergleichsoperand*					
	Abkürzung	*G*	*AM AA AB NA*	*NN FK*	*NL ZE*	*ID IG DU*	*ED EG*
Gruppe	*G*	a	a	a	a	a	a
Alphamerisches Datenelement	*AM*	a	a	a	a	x	a
Alphabetisches Datenelement	*AB*						
Aufbereitetes alphamerisches Datenelement	*AA*						
Aufbereitetes numerisches Datenelement	*NA*						
Nichtnumerisches Literal	*NN*	a	a	x	x	x	a
Figurative Konstanten mit Ausnahme von ZERO	*FK*						
Numerisches Literal	*NL*	a	a	x	x	n	n
Figurative Konstante ZERO	*ZE*						
Internes dezimales Datenelement	*ID*	a	x	x	n	n	n
Dualdatenelement	*DU*						
Internes Gleitpunktdatenelement	*IG*						
Externes dezimales Datenelement	*ED*	a	a	a	n	n	n
Externes Gleitpunktdatenelement	*EG*						

Bedeutung der Zeichen:

a	Nichtnumerischer Vergleich
n	Numerischer Vergleich
X	Vergleich ist nicht erlaubt

Abb. 18.23. Mögliche Vergleiche

18.23. Klassenbedingungen

Eine Klassenbedingung besitzt die Form:

bez IS [NOT] { NUMERIC / ALPHABETIC }

Die Angabe NUMERIC kann dann und nur dann niedergeschrieben werden, wenn unter bez der Name eines alphamerischen, externen dezimalen oder internen dezimalen Datenelementes verstanden wird. Die Niederschrift von ALPHABETIC dagegen setzt voraus, daß bez der Name eines alphabetischen oder alphamerischen Datenelementes ist.

Ein Datenelement gilt als alphabetisch, wenn sein Wert nur aus den 26 Buchstaben und dem Leerzeichen besteht. Ein Element, dessen Abbildung kein Abbildungszeichen für ein Vorzeichen enthält, wird als numerisch angesehen, wenn es nur aus den Ziffern 0 bis 9 besteht. Auch ein Element, dessen Abbildung auf eine Vorzeichenstelle verweist, gilt als numerisch, wenn es nur die Ziffern 0 bis 9 sowie zusätzlich ein mögliches Vorzeichen, d.h. damit die Buchstaben A bis R, auf der Einerstelle aufweist. Man beachte, daß ein einstelliges Datenelement mit einem Vorzeichen zur gleichen Zeit sowohl als numerisch als auch als alphabetisch gilt.

Beinahe alle COBOL-Programme müssen sich mit Plausibilitätsprüfungen von Eingabedaten herumschlagen. Die Klassenbedingungen ermöglichen eine einfache, obgleich oft unzulängliche Prüfung für allgemeine Zwecke. In Abb. 18.24. sind an einigen Beispielen die Ergebnisse solcher Klassenprüfungen dargestellt.

Wert des Datenelementes	*Numerische Klasse*	*Alphabetische Klasse*	*Nichtnumerische Klasse*	*Nichtalphabetische Klasse*
2345	W	F	F	W
234I	W	F	F	W
PU8Y	F	F	W	W
234S	F	F	W	W
R341	F	F	W	W
7	W	F	F	W
E	W	W	F	F
S	F	W	W	F
=	F	F	W	W
ABCD	F	W	W	F

Es bedeuten: W wahr; F falsch

Abb. 18.24. Beispiele für Klassenbedingungen und ihre Ergebnisse

18.24. Andere Bedingungen

In diesem Abschnitt wollen wir drei weitere Möglichkeiten kennenlernen, mit deren Hilfe wir Bedingungen formulieren können, die wir für IF-Statements benötigen.

1) *Vorzeichenbedingungen*

Vorzeichenbedingungen besitzen die Form:

ausdruck IS [NOT] { POSITIVE / NEGATIVE / ZERO }

Diese Vorzeichenbedingungen entsprechen den Vergleichsbedingungen (siehe Abschnitt 18.22.), wenn als Operator die Zeichen >, < oder gleich und als nachfolgender Ausdruck 0 gewählt wird, d.h. sie untersuchen, ob der Wert von ausdruck größer als, kleiner als oder gleich Null ist. Unter ausdruck muß stets ein numerischer Ausdruck erscheinen. Ein numerischer Wert ohne Vorzeichen gilt immer als positiv oder als Null.

2) *Bedingungen mit Bedingungsnamen*

Bedingungen mit Bedingungsnamen besitzen die Form:

[NOT] bedingungsname

Unter bedingungsname ist hierbei der Name irgendeines Datenelementes mit der Stufennummer 88 (siehe dazu Abschnitt 18.9.) zu verstehen. Die Niederschrift eines Bedingungsnamens ist gleichwertig einer Vergleichsbedingung, die den Wert des Datenelementes, auf den sich das Datenelement mit der Stufennummer 88 bezieht, mit dem Wert vergleicht, der für das Datenelement mit der Stufennummer 88 definiert wurde.

Die Bedingung wird als wahr erkannt, wenn das Datenelement den definierten Wert angenommen hat, im anderen Fall gilt falsch als Wert.

Runden wir die Betrachtungen über Bedingungsnamen durch das Beispiel in Abb. 18.25. ab.

Datenteil

```
Ø2  MONATE   PICTURE 99.
    88 JANUAR   VALUE IS 1.
    88 FEBRUAR   VALUE IS 2.
    88 DEZEMBER   VALUE IS 12.
```

Abb. 18.25. Beispiele für Bedingungen und Bedingungsnamen

Es brauchen nicht alle möglichen Werte des Datenelementes durch eine Eintragung auf der Stufe 88 erfaßt zu werden. Hätte bei unserem Beispiel MONATE den Wert 03 angenommen, dann würde das Statement

```
IF  FEBRUAR  ADD KONTO-ALT TO BETRAG GIVING KONTO-NEU.
```

ohne Wirkung bleiben. Wenn jedoch MONATE den Wert 02 besitzt, dann würde die Addition ausgeführt werden. Man beachte hierzu, daß Bedingungsnamenprüfungen je-

derzeit durch Vergleichsprüfungen unter Zugrundelegung des Wertes des Datenelementes ersetzt werden können. Somit ist

```
IF   FEBRUAR   ADD ···
```

gleichwertig der Anweisung

```
IF   MONATE = 2   ADD ···
```

3) *Bedingungen mit Schalterzuständen*

Im Betriebssystem DOS können zusätzlich auch Bedingungen abgefragt werden, die die Zustände von Schaltern (Schalterstati) betreffen. Ihre Form lautet:

```
[NOT] bedingungsname
```

Dabei müssen die hierzu eingesetzten Bedingungsnamen zuvor im Datenteil im Paragraphen SPECIAL-NAMES definiert sein (siehe auch Abschnitt 18.11.), wozu die Klausel

```
funktionsname  IS [mnemonischer-name]
                  { ON  } STATUS IS bedingungsname-1
                  { OFF }
                  { OFF } STATUS IS bedingungsname-2
                  { ON  }
```

dient.

In diese Vereinbarung ist an Stelle von funktionsname einer der folgenden Namen einzusetzen:

```
UPSI-Ø
UPSI-1
  :
  :
UPSI-7
```

Hierdurch wird eine Bezugnahme auf die acht Bitstellen des UPSI-Bytes erreicht, das durch das UPSI-Statement der Jobsteuersprache eingerichtet wird. Es kann jeweils nur ein Bedungungsname für den Ein-Zustand und für den Aus-Zustand des Bits eingeführt werden. Diese Bedingungsnamen sind gleichwertig den Eintragungen von Datenelementen mit der Stufennummer 88, die dem Datennamen mnemonischer-name oder dem äquivalenten Namen UPSI-n mit n=0,1,···,7 untergeordnet sind. Wenn erforderlich, können die Bedingungsnamen gekennzeichnet sein, wobei entweder die Bezeichnung mnemonischer-name oder UPSI-n benutzt werden darf.

Da Bedingungen über Schalterzustände nur im Betriebssystem DOS abgefragt werden können, sind Statements, die solche Bedingungen enthalten, nicht verträglich mit dem Betriebssystem OS. Es ist daher dringend anzuraten, derartige Statements generell zu vermeiden.

18.25. Verbundene Bedingungen

Jeder Bedingung kann das Wort NOT vorausgestellt werden. Es bewirkt, daß der Wahrheitsgehalt der betreffenden Bedingung gerade umgekehrt wird.

Zwei oder mehrere Bedingungen können zu einer Verbundbedingung zusammengefaßt werden, indem man sie mit Hilfe der logischen Operatoren AND oder OR miteinander verknüpft. Verbundbedingungen besitzen infolgedessen folgendes Aussehen:

```
[NOT] bed-1 { {AND} [NOT] bed-2 ··· }
            { {OR }
```

Jede Einfachbedingung bed-i muß hierbei von der nachfolgenden Einfachbedingung durch AND oder OR getrennt sein, das sei noch einmal grundsätzlich betont. Wird AND niedergeschrieben, so gilt die zusammengefaßte Bedingung dann und nur dann als wahr, wenn beide Einfachbedingungen auf wahr führen. Bei Benutzung des logischen Operators OR führt die Verbundbedingung immer dann auf wahr, wenn irgendeine Einfachbedingung oder wenn beide miteinander verknüpften Einfachbedingungen wahr sind.

Wenn aufeinanderfolgende Bedingungen Vergleichsbedingungen mit dem gleichen Subjekt [1] (ersten Operanden) sind, kann das Subjekt von alle Folgebedingungen eliminiert werden, jedoch nicht bei der ersten. Somit darf die Verbundbedingung

```
A > B  OR  A = C  AND  A = D
```

auch kurz wie folgt geschrieben werden:

```
A > B  OR  = C  AND  = D
```

Ist auch der Vergleichsoperator bei solchen aufeinanderfolgenden Bedingungen gleich, so darf er ebenfalls weggelassen werden. Somit kann

```
A = E  OR  A = F  AND  A = C
```

auch zu

```
A = E  OR  F  AND  C
```

geschrieben werden.

Wird NOT benutzt und ist das Subjekt weggelassen, so wird dieses Wort grundsätzlich auf das Subjekt bezogen und nicht als Teil des Vergleichsoperators betrachtet. Somit wird

```
A = H  AND  NOT I  OR  J
```

zu

```
(A = H)  AND  NOT (A = I)  OR  (A = J)
```

interpretiert.

Klammern können zur Einschließung von Bedingungen genau so verwendet werden wie in arithmetischen Ausdrücken. Wird durch Klammern nicht die Reihenfolge der Auswertung von Bedingungen komplett angezeigt, so gilt folgende Reihenfolge:

1) Arithmetische Ausdrücke
2) Vergleichsoperationen
3) NOT
4) AND (stets von links nach rechts)
5) OR (stets von links nach rechts)

Zur Illustration wollen wir ein Beispiel betrachten. Es möge lauten:

```
A + K = L  AND  NOT M > L
```

Die Auswertung erfolgt so, als ob folgendes niedergeschrieben wäre:

```
((A + K) = L)  AND  ((NOT (M > L))
```

Die Regeln von COBOL stimmen also mit den normalen Regeln der Logik und Mathematik für die Auswertung logischer Ausdrücke überein.

---

1) Unter dem Subjekt versteht man hier den Teil einer Vergleichsbedingung, der links vom Vergleichsoperator steht.

Aufgabengruppe 50

1) Es wurde die nachstehende Statementfolge ausgeführt:

```
IF  A > B  IF  A   C  ADD B TO C
                      ELSE  ADD A, B GIVING C
           ELSE  IF  B > C  ADD A TO C
                            ELSE  ADD A, B GIVING C.
```

Welcher Wert ergibt sich für C, wenn folgende Anfangswerte vorliegen?

a) A=1, B=2, C=4
b) A=1, B=4, C=2
c) A=2, B=1, C=4
d) A=2, B=4, C=1
e) A=4, B=1, C=2
f) A=4, B=2, C=1
g) A=2, B=2, C=2

2) Es wurde folgender Satz ausgeführt:

```
MOVE 4 TO C;
IF  A ZERO  THEN  MOVE 2 TO C
IF  B POSITIVE  MOVE 3 TO C
                ELSE  MOVE 5 TO C.
```

Welcher Wert ergibt sich für C, wenn folgende Anfangswerte vorliegen?

a) A=Ø, B=1
b) A=Ø, B=-1
c) A=1, B=-1
d) A=1, B=Ø
e) A=-1, B=1

3) Es wird ein Kartenstapel eingelesen, auf dessen einzelnen Karten in den Spalten 1 bis 4 römische Ziffern abgelocht sind. Es werden nur die Ziffern I, V, X und L benutzt.

Schreibe ein COBOL-Programm, das die römischen Eingabeziffern und ihre Äquivalente des Dezimalsystems druckt!

4) Das Programm zu Aufgabe 3) ist wesentlich zu erweitern. Die römischen Ziffern belegen jetzt die Spalten 1 bis 8; zusätzlich können auch C, D und M auftreten. Die römischen Ziffern bilden zusammen eine Zahl. Neben den römischen Ziffern ist die äquivalente Dezimalzahl zu drucken. Die römischen Ziffern belegen nicht notwendig alle 8 Kartenspalten; rechts von der Zahl können innerhalb des achtspaltigen Feldes leere Kartenspalten vorhanden sein.

5) Schreibe ein COBOL-Programm, mit dem eine Karte gelesen werden kann, die auf den Spalten 1 bis 6 ein Datum in der Form ttmmjj enthält (tt Tag, mm Monat, jj Jahr). Ein gültiges Datum soll in die Form jjttt umgewandelt werden, wobei unter ttt der laufende Tag des Jahres verstanden werden soll (1 bis 365 bzw. 366 bei Schaltjahren). Das Eingabedatum und das errechnete Datum ist zu drucken.

6) Die Aufgabe 5) ist wie folgt zu ergänzen: Es ist für jedes eingelesene Datum zwischen 1975 und 1988 außerdem der Wochentag zu bestimmen und zu drucken.

19. Sprachelemente von COBOL für den Fortgeschrittenen

19.1. Die Option CORRESPONDING

In drei verschiedenen Statements kann die Option CORRESPONDING benutzt werden.

1) MOVE CORRESPONDING bez-1 TO bez-2

2) ADD CORRESPONDING bez-1 TO bez-2 [ROUNDED]
[ON SIZE ERROR imperativanweisung]

3) SUBTRACT CORRESPONDING bez-1 FROM bez-2 [ROUNDED]
[ON SIZE ERROR imperativanweisung]

Ihre Wirkungsweise ähnelt der der gleichen Statements ohne CORRESPONDING-Option. An Stelle von CORRESPONDING darf auch die Abkürzung CORR benutzt werden. Sowohl bez-1 als auch bez-2 stellen Bezeichnungen dar, die auf Gruppen verweisen.

Durch die Ausführung der ADD- bzw. der SUBTRACT-Statements werden die Datenelemente mit den gleichen Namen, die innerhalb der durch bez-1 bzw. bez-2 bezeichneten Gruppen vorkommen, additiv bzw. subtraktive miteinander verknüpft. Datenelemente, deren Namen nicht zueinander passen, werden von der Operation nicht berührt. Die Bedingung SIZE ERROR tritt nach Ausführung der letzten Einzeloperation dann auf, wenn es bei irgendeiner Addition oder Subtraktion zu einem Überlauffehler gekommen ist. Operationen, bei denen ein solcher Fehler festgestellt wurde, werden wie üblich unterdrückt.

Bei der Ausführung eines MOVE-Statements werden die Werte derjenigen Datenelemente, die durch die OCCURS-, REDEFINES- oder RENAMES-Klausel ausgezeichnet sind, nicht übertragen. Die Werte der Elemente, die solchen mit einer derartigen Klausel versehenen Elementen untergeordnet sind, werden jedoch übertragen.

Gekennzeichnete Namen dürfen verwendet werden. Zu übertragende, zu addierende und zu subtrahierende Elemente gelten als paarweise vorliegend, wenn die Kennzeichnung ihrer Namen bis herauf zu bez-1 und bez-2, diese jedoch ausgenommen, die gleiche ist. Sowohl bez-1 als auch bez-2 selbst können gekennzeichnet sein.

Die Grundidee zur Einführung der CORRESPONDING-Option bestand darin, mehrere Elemente auf einfache Art und Weise operativ zu verknüpfen, die in zwei verschiedenen Gruppen den gleichen Namen besitzen. Es wird dadurch auf eine bequeme Methode die Spezifizierung einer Operationsfolge durch ein einziges Statement erreicht.

19.2. Das Statement EXAMINE

In zwei Formen kann das EXAMINE-Statement auftreten. Sie lauten:

a)
```
EXAMINE bez TALLYING { ALL       } lit-1
                     { LEADING   }
                     { UNTIL FIRST }
                [REPLACING { BY lit-2] }
```

b)
```
EXAMINE bez REPLACING { ALL       } lit-1 BY lit-2
                      { LEADING   }
                      { UNTIL FIRST }
                      { FIRST     }
```

Stellt bez den Namen eines nichtnumerischen Datenelementes dar, so müssen sowohl lit-1 als auch lit-2 einstellige nichtnumerische Literale vom gleichen Datentyp wie bez sein. Auch figurative Konstanten (mit Ausnahme von ALL) dürfen an Stelle der Literale benutzt werden. Ist bez dagegen der Name eines numerischen Datenelementes, so muß dieses mit DISPLAY in der Verwendungsklausel erklärt sein; die Literale wiederum müssen vorzeichenfreie ganze Zahlen zwischen 0 und 9 sein. An Stelle der Zahl 0 darf auch die figurative Konstante ZERO benutzt werden. Jedes Vorzeichen, das in der Abbildung von bez definiert ist, wird ignoriert.

In allen Fällen wird der Wert von bez von links nach rechts durchsucht.

Bei der Niederschrift von REPLACING geschieht bei der Ausführung des EXAMINE-Statements folgendes:

1) Wenn ALL angegeben ist, so ersetzt das durch lit-2 festgelegte Zeichen alle in bez auftretenden Zeichen lit-1.

2) Bei Angabe von LEADING wird die Ersetzung der Zeichen lit-1 durch die Zeichen lit-2 in dem Augenblick beendet, in dem im angesprochenen durch bez benannten Feld das erste von lit-1 verschiedene Zeichen festgestellt wird.

3) Wenn UNTIL FIRST niedergeschrieben wird, so ersetzt lit-2 jedes andere Zeichen des durch bez benannten Feldes solange, bis das erste Zeichen lit-1 auftritt. Erscheint im Feld kein solches Zeichen, so werden alle Zeichen des Feldes durch lit-2 ersetzt.

4) Bei Angabe von FIRST wird nur das erste Zeichen lit-1 durch das Zeichen lit-2 ersetzt.

Wird TALLYING niedergeschrieben, so wird ein Zähler eingerichtet, der die Anzahl der ersetzten Zeichen im speziellen Register TALLY zählt. Dieses Register besitzt die Standardabbildung S9(5) und die Verwendungsklausel COMPUTATIONAL. Es darf überall dort in einem Programm benutzt werden, wo die Verwendung eines Datennamens gestattet ist. Wird die Klausel REPLACING BY weggelassen, so wird zwar der Zähler gesetzt, der Wert von bez jedoch nicht geändert. Nehmen wir z.B. einmal an, daß ein Datenfeld mit Namen A die Zeichenfolge DDL7D enthält. Wenn nun das Statement

```
EXAMINE A TALLYING ALL 'D'.
```

ausgeführt wird, so wird TALLY der Wert 3 zugewiesen. Das Statement

```
EXAMINE A TALLYING UNTIL FIRST 'D'.
```

setzt dagegen TALLY auf den Wert 0. Das Statement

```
EXAMINE A REPLACING ALL 'D' BY 'E'.
```

bewirkt, daß nach seiner Ausführung im Feld A die Zeichenfolge EEL7E steht. Die Anweisung

```
EXAMINE A REPLACING FIRST 'D' BY 'E'.
```

führt schließlich dazu, daß nach ihrer Ausführung das Feld A die Zeichenfolge EDL7D enthält.

19.3. Das Statement TRANSFORM

Das Statement TRANSFORM besitzt folgende Gestalt:

TRANSFORM bez-3 FROM bez-1 TO bez-2

Hierbei ist bez-3 entweder der Name einer Gruppe oder der Name eines nichtnumerischen Datenelementes. Sowohl bez-1 als auch bez-2 stellen entweder nichtnumerische Literale oder die Namen von Gruppen oder von alphabetischen oder alphamerischen Datenelementen dar. An Stelle nichtnumerischer Literale dürfen auch figurative Konstanten mit Ausnahme von ALL benutzt werden; bei ihrer Verwendung werden sie jedoch als einstellige Größen betrachtet.

Wenn sowohl bez-1 als auch bez-2 Datennamen darstellen, so müssen durch diese Felder gleicher Länge angesprochen werden. In allen anderen Fällen muß die Länge von bez-2 entweder gleich einem Byte sein oder sonst gleich der von bez-1 sein. Für bez-1 ist eine maximale Länge von 255 Zeichen erlaubt.

Besitzen bez-1 und bez-2 die gleiche Länge, so wird jedes Zeichen, das in dem durch bez-3 bezeichneten Feld auftritt und gleich einem der Zeichen ist, die im durch bez-1 bezeichneten Feld vorhanden sind, durch das Zeichen ersetzt, das in dem durch bez-2 bezeichneten Feld an der gleichen Stelle steht wie das zu ersetzende Zeichen. Besitzt das durch bez-2 bezeichnete Feld die Länge 1, so wird jedes Zeichen, das im Feld bez-3 auftaucht und gleich einem der in bez-1 vorhandenen Zeichen ist, durch das in bez-2 stehende Zeichen ersetzt.

Einige Beispiele mögen dieses Statement illustrieren:

1)
```
MOVE '35ØS5L' TO Y.
TRANSFORM Y FROM '35' TO '48'.
```

Als Ergebnis des TRANSFORM-Statements ist im Feld Y der Wert '48ØS8L' vorhanden.

2)
```
MOVE '35ØS5L' TO Y.
TRANSFORM Y FROM '35' TO 'E'.
```

Diese Satzfolge führt auf den Wert 'EEØSEL' für Y.

3) Eine Reihe von TRANSFORM-Ergebnissen ist in Abb. 19.1. erfaßt.

Ausgangsfeld	FROM	TO	Zielfeld
ØØ179Ø	ZERO	SPACE	bb179b
ØØ179Ø	'189'	'264'	ØØ274Ø
ØØ179Ø	'189'	'5'	ØØ575Ø
7,324.	'.,'	',.'	7.324,

Anmerkung: b bedeutet hier das Leerzeichen

Abb. 19.1. Beispiele für das TRANSFORM-Statement

Der Vorteil, für bez-1 und bez-2 Datennamen zu benutzen, besteht augenscheinlich darin, daß die in die Transformation, d.h. die in die Übersetzung eingehenden Zeichen, noch während der Programmausführung geändert werden können.

19.4. Das Statement PERFORM

Das PERFORM-Statement bewirkt, daß eine Prozedur oder eine Prozedurmenge außerhalb der normalen Reihenfolge ausgeführt wird; nach ihrer Ausführung erfolgt die Rückkehr zu dem Statement, das auf das PERFORM-Statement folgt.

Das PERFORM-Statement weist verschiedene Formen auf, die einfachste ist die folgende:

```
PERFORM proz-1 [THRU proz-2]  [bez TIMES]
                              [UNTIL bed]
```

Hierbei bedeuten proz-1 bzw. proz-2 die Namen von Prozeduren, d.h. die Namen von Paragraphen oder von Kapiteln, bei bez handelt es sich um ein numerisches Datenelement oder um ein numerisches Literal und schließlich ist durch bed eine Bedingung gemeint.

Bei der Ausführung der PERFORM-Anweisung geschieht folgendes: Die Programmablaufsteuerung wird zunächst an das erste Statement der Prozedur proz-1 übertragen. Danach wird der Programmablauf in normaler Weise gesteuert bis schließlich das letzte Statement in proz-2 (oder in proz-1, wenn die THRU-Klausel nicht spezifiziert ist) ausgeführt ist. Zu diesem Zeitpunkt erfolgt die Rückgabe der Programmablaufsteuerung an das auf das PERFORM-Statement folgende Statement. Ist die TIMES-Klausel hinzugefügt, so werden die Prozeduren, auf die Bezug genommen wird, so oft ausgeführt, wie es der durch bez angesprochene Wert verlangt; erst dann erfolgt die Rückkehr. Besitzt jedoch bez einen nichtpositiven Wert, so erfolgt keine Ausführung der bezeichneten Prozeduren und die Programmablaufsteuerung geht unmittelbar zu dem auf das PERFORM-Statement folgende Statement über. Erfolgt die Codierung der Klausel UNTIL, so wird zunächst die unter bed genannte Bedingung ausgewertet. Die Prozeduren gelangen nur dann zur Ausführung, wenn die Bedingung falsch ist. Nach jeder Ausführung der Prozeduren kommt es zur erneuten Auswertung der Bedingung. Von dieser hängt dann der wiederholte Ablauf der Prozeduren ab. Wird für bed der Wert „wahr" gefunden, so unterbleibt die Ausführung der Prozeduren.

Stellt bez einen Datennamen dar, so kann der Wert des durch ihn repräsentierten Datenelementes während des Ablaufes der Prozeduren geändert werden. Jedoch, eine Tatsache muß festgehalten werden: Die Anzahl der Prozedurwiederholungen ist durch den Wert von bez bestimmt, der zur Ausführungszeit des PERFORM-Statements vorliegt, d.h. vor Aufruf der Prozeduren.

Wegen der Notwendigkeit, daß die Programmablaufsteuerung auch wirklich bis zum Ende von proz-2 gehen muß, ist das EXIT-Statement geschaffen worden. Seine Form lautet:

```
pn. EXIT
```

Unter pn ist der Name eines Paragraphen zu verstehen. Der zu diesem Paragraphen gehörende Satz darf im Paragraphen enthalten sein. Während der Ausführung eines Programmes bewirkt ein EXIT-Statement keine Operation. Es kann daher als bloßer Bezugspunkt dienen, beispielsweise als Sprungziel für eine unbedingte Verzweigungsanweisung GO TO.

Innerhalb der Prozeduren, die durch ein PERFORM-Statement aktiviert werden, dürfen auch die Statements GO TO und PERFORM selbst codiert werden, allerdings nur bei Beachtung gewisser Einschränkungen. Ein PERFORM-Statement muß sich nämlich auf Prozeduren beziehen, die entweder gänzlich außerhalb oder völlig innerhalb des Wirkungsbereiches des ersten PERFORM-Statements liegen. Sie dürfen allerdings gemeinsam einen gewöhnlichen EXIT-Paragraphen haben. Eine Sprunganweisung GO TO darf

nicht eine Verzweigung zu einem Punkt außerhalb von proz-1 beinhalten, wenn die THRU-Klausel fehlt. Bei Benutzung einer unbedingten Verzweigungsanweisung muß stets daran gedacht werden, daß sie unter allen Umständen ihr Sprungziel innerhalb der durch das PERFORM-Statement benannten Prozeduren haben muß, d.h. das letztmögliche Sprungziel ist das Ende der Prozeduren.

Wird ein PERFORM-Statement durch ein anderes aufgerufen, so sagt man von den Prozeduren, daß sie ineinander verschachtelt oder eingebettet sind.

Normalerweise wird so programmiert, daß die von einem PERFORM-Statement aufgerufenen Prozeduren abseits vom Rest des Prozedurteils gestellt und somit nicht in normaler Reihenfolge ausgeführt werden. Es ist jedoch ohne weiteres möglich, daß die Prozeduren dadurch zum Ablauf gebracht werden, indem sie durch eine unbedingte Verzweigungsanweisung angesprungen werden bzw. einfach dadurch, daß sie von der Programmablaufsteuerung in linearer Verfolgung des Programmweges betreten werden. Solche Programme neigen zu einer gewissen Unübersichtlichkeit, so daß bei ihnen die Fehlersuche erschwert ist. Im allgemeinen sollte man deshalb ein solches Vorgehen in der Programmierung vermeiden.

Um das PERFORM-Statement zu veranschaulichen, wollen wir ein Beispiel betrachten. Nehmen wir dazu an, daß N, X, Y und FEH als Datenelemente mit COMPUTATIONAL-1 definiert wurden. Die nachfolgenden Anweisungen dienen zur Berechnung der Quadratwurzel von N und speichern dieselbe unter Y; die Codierung ist in Abb. 19.2. dargestellt.

```
SQRT SECTION.
SQRT1.
    MOVE 1 TO X.
SQRT2.
    COMPUTE   Y = Ø.5 * X ' N / (2 * X).
    IF   X > Y   THEN COMPUTE   FEH = X - Y,
                 ELSE COMPUTE   FEH = Y - X.
    IF   FEH > Ø.ØØ5,   MOVE Y TO X,   GO TO SQRT2.
```

Abb. 19.2. Statementfolge zur Berechnung von Quadratwurzeln $y = \sqrt{n}$

Haben wir nun beispielsweise die Quadratwurzeln von W_1, W_2 und W_3 zu berechnen, so können wir uns der Statementfolgen bedienen, die in Abb. 19.3. dargestellt sind.

```
MOVE W1 TO N.   PERFORM SQRT.   MOVE Y TO WSQRT1.
MOVE W2 TO N.   PERFORM SQRT.   MOVE Y TO WSQRT2.
MOVE W3 TO N.   PERFORM SQRT.   MOVE Y TO WSQRT3.
```

Abb. 19.3. Beispiel zur Berechnung von Quadratwurzeln unter Ausnutzung der in Abb. 19.2. vorgestellten Prozedur

Das PERFORM-Statement kann auch in einer zweiten Form benutzt werden. Diese lautet:

```
PERFORM proz-1   [THRU proz-2]
        VARYING bez-1 FROM bez-2 BY bez-3 UNTIL bed-1
       [AFTER bez-4 FROM bez-5 BY bez-6 UNTIL bed-2]
       [AFTER bez-7 FROM bez-8 BY bez-9 UNTIL bed-3 ]
```

Hierbei sind proz-1 und proz-2 die Namen von Prozeduren, d.h. von Paragraphen oder Kapiteln. Unter bed-1, bed-2 und bed-3 sind Bedingungen zu verstehen. Schließlich

bedeuten bez-1, bez-4 und bez-7 die Namen numerischer Datenelemente; die übrigen Bezeichnungen bez-2, bez-3, bez-5, bez-6, bez-8 und bez-9 verweisen auf numerische Literale oder auf numerische Datenelemente.

Bei Entdeckung dieses Statements werden vom Kompilierer im Objektprogramm drei temporäre Datenfelder eingerichtet. Wir wollen sie einmal tf-2, tf-5 und tf-8 nennen; ihre Abbildungen sind die gleichen wie die von bez-2, bez-5 und bez-8. Das PERFORM-Statement in der o.a. Form wird dann unter Verwendung der zeitweiligen Felder so ausgeführt, als ob wir die in Abb. 19.4. dargestellte Statementfolge codiert hätten.

	Schritt	*Statementfolge*
	1	MOVE bez-2 TO tf-2 MOVE bez-5 TO tf-5 MOVE bez-8 TO tf-8.
	2	MOVE tf-2 TO bez-1 MOVE tf-5 TO bez-4 MOVE tf-8 TO bez-7.
P3	3	IF bed-1 THEN GO TO P8.
P4	4	IF bed-3 THEN MOVE tf-5 TO bez-4 ADD bez-3 TO bez-1 GO TO P3.
P5	5	IF bed-3 THEN MOVE tf-8 TO bez-7 ADD bez-6 TO bez-4 GO TO P4.
	6	PERFORM proz-1 [THRU proz-2].
	7	ADD bez-9 TO bez-7 GO TO P5.
P8	8	EXIT.

Abb. 19.4. Auflösung der PERFORM-Statements mit der VARYING-Option durch den Kompilierer

Es erscheint uns unumgänglich notwendig, die Abb. 19.4. durch einige Bemerkungen zu ergänzen. Wenn die zweite AFTER-Klausel fehlt, so gelten die Statements, in die bez-7, bez-8, bez-9 und tf-8 eingehen, als gestrichen. Das gleiche gilt für bez-4, bez-5, bez-6 und tf-5, wenn auch die erste AFTER-Klausel weggelassen ist.

Es ist zu beachten, daß Änderungen, die mit bez-2, bez-5 und bez-8 vorgenommen werden, die Ausführungsanzahl der Prozeduren von proz-1 (bis evtl. zu proz-2) nicht berühren. Diese bleibt in der Zahl erhalten, wie sie beim Starten des PERFORM-Statements eingerichtet wurde.

Die Betrachtungen über das PERFORM-Statement in der erweiterten Form sollen durch ein Beispiel abgeschlossen werden. Der Auszug aus dem Datenteil des Beispielprogrammes ist in Abb. 19.5. dargestellt, der Auszug aus dem Prozedurteil in Abb. 19.6.

```
77 I  COMPUTATIONAL PICTURE S9.
77 J  COMPUTATIONAL PICTURE S9.
77 K  COMPUTATIONAL PICTURE S9.
77 SM PICTURE S99999.
Ø1 FILLER.
   Ø2 FILLER OCCURS 2 TIMES.
      Ø3 FILLER OCCURS 3 TIMES.
         Ø4 Q OCCURS 2 TIMES PICTURE S99.
```

Abb. 19.5. Auszug aus dem Datenteil des Beispielprogrammes

```
MOVE ZEROS TO SM.
PERFORM S VARYING I FROM 1  UNTIL I > 2
          AFTER J FROM 3 BY -1  UNTIL J < Ø
          AFTER K FROM 1 BY 1  UNTIL K > 2.
```

Hierbei lautet die mit S benannte Prozedur wie folgt:

```
S.  ADD  Q (I, J, K) TO SM.
```

Abb. 19.6. Auszug aus dem Prozedurteil des Beispielprogrammes

Die Statements in Abb. 19.6. bewirken, daß die Elemente von Q in der nachstehenden Reihenfolge zur Summe SM addiert werden:

```
Q(1,3,1)
Q(1,3,2)
Q(1,2,1)
Q(1,2,2)
Q(1,1,1)
Q(1,1,2)
Q(2,3,1)
Q(2,3,2)
Q(2,2,1)
Q(2,2,2)
Q(2,1,1)
Q(2,1,2)
```

Da die Prozeduren durch die PERFORM-Anweisung gewissermaßen direkt aufgerufen werden, besteht auch kein Grund, daß sie innerhalb der natürlichen Reihenfolge eines Programmes, die von Paragraph zu Paragraph läuft, erscheinen müssen. Es ist vielmehr geradezu üblich, die Prozeduren, die durch PERFORM-Anweisungen angesprochen werden, außerhalb des normalen Programmablaufes niederzuschreiben. Es ist erlaubt, daß innerhalb einer Prozedur, die durch eine PERFORM-Anweisung aufgerufen wird, weitere PERFORM-Anweisungen stehen können. Es ist nur erforderlich, daß das Ende einer so aufgerufenen Prozedur nicht inmitten der Statements einer anderen durch PERFORM aufgerufenen Prozedur liegt. Die durch PERFORM aufgerufenen Prozeduren müssen in der umgekehrten Reihenfolge abgeschlossen werden, wie sie aufgerufen wurden.

19.5. Die SIGN-Klausel im Datenteil

Im Datenteil ist die Niederschrift der Klausel

SIGN IS {LEADING / TRAILING} [SEPARATE CHARACTER]

für Gruppen und für numerische Datenelemente, die mit DISPLAY in der Verwendungsklausel erklärt sind, erlaubt; Voraussetzung hierzu ist freilich, daß die Abbildung ein Abbildungszeichen S enthält. Wird die SIGN-Klausel bei einem Satzbestandteil angegeben, der eine Gruppe darstellt, so gilt sie für jedes numerische Datenelement in dieser Gruppe, deren Abbildung ein S aufweist.

Fehlt SEPARATE, so nimmt der Kompilierer an, daß das Vorzeichen ein Überstellenvorzeichen ist, das auf der ersten oder auf der letzten Ziffernstelle des Zahlenfeldes erscheint. Fehlt die gesamte SIGN-Klausel, so ist die Wirkung die gleiche, wie wenn

```
SIGN IS TRAILING
```

niedergeschrieben wird.

Bei Codierung von SEPARATE wird angenommen, daß Abbildungszeichen S in der Abbildung eine gesonderte Stelle (ein gesondertes Byte) belegt, und zwar entweder unmittelbar vor der ersten oder unmittelbar nach der letzten Ziffernstelle. Während der Programmausführung muß dann diese Position ein Pluszeichen oder ein Minuszeichen enthalten. Das Fehlen eines Vorzeichens hat u. U. die anormale Beendigung eines Programmes zur Folge.

Die Kompilierer der Version 2 unterstützen die SIGN-Klausel nicht.

19.6. Die Formatsteuerung

Gewisse Statements beeinflussen das übersetzte Programm nicht. Die sorgen nur dafür, daß die Auflistung des Programmes übersichtlicher lesbar und damit einfacher zu handhaben sind. Die in diesem Abschnitt zu behandelnden Statements sind von dieser Natur. Sie müssen vollständig innerhalb der Randbegrenzung B geschrieben werden und dürfen auf einer Zeile nur als einzige Statements erscheinen. Die Stellen 8 bis 11 der Zeile (Randbegrenzung A) müssen leer bleiben. Ein abschließender Punkt ist nicht gestattet.

Das Statement

```
EJECT
```

bewirkt, daß der Kompilierer zu einer neuen Seite übergeht, wenn er mit dem Auflisten des Quellenprogrammes beschäftigt ist. Die Statements

```
SKIP1
SKIP2
SKIP3
```

haben hingegen zur Folge, daß 1, 2 bzw. 3 Leerzeilen erzeugt werden, wenn das Quellenprogramm aufgelistet wird.

19.7. Bibliothek mit Quellenstatements

Wenn eine Statementfolge oder eine Dateibeschreibung in mehreren Programmen einzusetzen ist, so kann sicherlich die Methode, diese Statementfolge oder diese Dateibeschreibung in eine Bibliothek auf einem Plattenspeicher zu stellen, als beste bezeichnet werden. Jedes Programm, in dem die Statementfolge oder die Dateibeschreibung benötigt wird, kann sich dann des Eintrags in der dafür angelegten Bibliothek bedienen. Diese Methode gewährleistet darüberhinaus, daß alle Programme stets die neueste Version der Codierung benutzen und daß erfolgte Änderungen der Codierung in allen Programmen gleichermaßen berücksichtigt werden, nachdem sie neu übersetzt sind.

Im Betriebssystem DOS müssen die für COBOL bestimmten Bibliotheksteile Eintragungen aufweisen, die mit dem Buchstaben C beginnen, dem ein Punkt folgt. Diesem Punkt wiederum müssen sich 1 bis 8 Zeichen anschließen, die den Regeln für die Bildung von externen COBOL-Namen genügen. Wenn ein Kompilierer nach einer Eintragung, beispielsweise nach PRUEFG, sucht, so bildet er zunächst den Buchnamen, hier C.PRUEFG. Danach wird die Bibliothek SYSSLB, sofern dem System zugewiesen, nach einem Buch mit dem gebildeten Namen durchforscht. Wird keine gefunden, dann wird die Quellenstatementbibliothek auf SYSRES ebenfalls durchsucht.

Im Betriebssystem OS ist in der Jobsteuersprache ein mit dem Namen SYSLIB versehenes DD-Statement zu formulieren. Während der Übersetzung dient dieses dazu, die Quellenstatementbibliothek zu identifizieren (d.h. eine untergliederte Datenmenge, die für die Übersetzung heranzuziehen ist). Das DD-Statement

```
//COB.SYSSLB   DD   DISP=SHR,DSNAME=···
```

kommt unmittelbar vor das DD-Statement, das auf COB.SYSIN bezugnimmt, wenn die von der IBM herausgegebenen Standardprozeduren benutzt werden.

Mit Ausnahme von Satzbeschreibungen (Eintragungen mit den Stufennummern 01 bzw. 77) ist in die Bibliothek der gesamte Code aufzunehmen, der später den vorgesehenen Platz im Quellenprogramm einnehmen soll. Bei Eintragungen mit den Stufennummern 01 und 77 kann zwar die Eintragung mit der Stufennummer anfangen, der dann ein Datenname folgen kann. Doch sei darauf verwiesen, daß Stufennummer und Datenname nicht von der Bibliothek ins Quellenprogramm übernommen werden. Beide können jedoch zu Dokumentationszwecken dienen; ihre Anwesenheit erleichtert die Übersicht über die in die Bibliothek hineingestellten Informationen.

Das Statement

```
COPY bibln [SUPPRESS]
           [REPLACING { wort-1 BY { wort-2    } } ···]
                      {           { literal-2 } }
                      {           { bez-2     } }
```

dient dazu, eine Eintragung aus der Quellenstatementbibliothek ins Programm zu übernehmen. Die Einfügung erfolgt an der Programmstelle, an der das COPY-Statement erscheint. Die COPY-Statements dürfen allerdings nur an den Stellen fixiert werden, die in der Abb. 19.7. kenntlich gemacht sind.

```
SOURCE-COMPUTER. COPY text
OBJECT-COMPUTER. COPY text
SPECIAL-NAMES. COPY text
FILE-CONTROL. COPY text
I-O-CONTROL. COPY text
SELECT dateiname COPY text
FD dateiname COPY text
Ø1 datenname COPY text
77 datenname COPY text
kapitelname SECTION. COPY text
paragraphenname. COPY text
```

Abb. 19.7. Die Stellung von COPY-Statements in einem Programm

Die ersten acht Zeichen von bibln bezeichnen den Namen, unter dem die Eintragung in die Bibliothek aufgenommen wurde. So würde z.B. durch

```
COPY PRUEFG
```

veranlaßt werden, daß im Betriebssystem DOS nach C.PRUEFG gesucht wurde bzw. im Betriebssystem OS nach einem Glied namens PRUEFG. Wird SUPPRESSED geschrieben, so werden die eingefügten Beschreibungen bzw. Statements nicht gedruckt, wenn die Programmauflistungen ausgegeben werden. Wenn die REPLACING-Klausel benutzt wird, so wird wort-1, welches irgendein beliebiges nichtreserviertes gültiges COBOL-Wort sein kann, durch die Größe ersetzt, die hinter BY angegeben ist, und zwar überall dort in den kopierten Beschreibungen bzw. Statements, wo es auftritt. Unter bez-2 kann auch ein gekennzeichneter Datenname aufgeführt sein. Innerhalb des COPY-Statements können beliebig viele Ersetzungen veranlaßt werden.

Das COPY-Statement muß immer als letztes Statement einer Zeile des Quellenprogramms codiert sein. Mit Ausnahme des Paragraphen FILE-CONTROL kann keine andere Klausel und kein anderes Statement in die Eintragung aufgenommen werden, die das COPY-Statement enthält.

Der Datenname mit der Stufennummer 01 bzw. 77 tritt automatisch an die Stelle des in der Bibliothek gespeicherten Datennamens. Diese Ersetzung erfolgt nur für Eintragungen mit den beiden genannten Stufennummern. Dort, wo eine andere Ersetzung innerhalb des zu kopierenden Textes vorgenommen werden soll, muß man sich generell der REPLACING-Klausel bedienen.

Aufgabengruppe 51

1) Im Datenteil eines Programmes liegt eine Erklärung des Datenelementes E vor, als Abbildung ist X(250) vorgesehen.

   Durch Eingabe ist E ein Text zugewiesen worden, der aus Großbuchstaben, Ziffern und Sonderzeichen besteht. Schreibe eine Statementfolge für den Prozedurteil nieder, die dafür sorgt, daß die Großbuchstaben in Groß- und Kleinbuchstaben übergeführt werden. Alle Buchstaben mit Ausnahme des ersten Buchstabens eines Satzes sind in Kleinbuchstaben umzuwandeln. Als Satzende fungiert ein Punkt, sofern er von einem oder mehreren Leerzeichen gefolgt ist. - Es sind außerdem die erforderlichen Eintragungen für das Arbeitsspeicherkapitel im Datenteil niederzuschreiben.

2) Im Datenteil eines Programmes liegen die beiden folgenden Erklärungen vor:

```
xx  F  COMPUTATIONAL-3  PICTURE S9(3)  OCCURS 1Ø.
xx  G  COMPUTATIONAL-3  PICTURE S9(2Ø).
```

Unter xx sei hierbei eine beliebige gültige Stufennummer verstanden. Codiere die Erklärungen für das Arbeitsspeicherkapitel und die Statements für den Prozedurteil, mit deren Hilfe G das Produkt aus allen um 34 verminderten Bereichselementen von F zugewiesen wird, d.h. es gilt

$$g = \prod_{i=1}^{10} f_i - 34$$

3) Die Aufgabe 2 ist wie folgt abzuändern: Für F liegt die Klausel

```
OCCURS 1Ø DEPENDING ON J
```

vor. Dabei ist J mit der Abbildung S99 und mit COMPUTATIONAL erklärt. Bilde unter dieser Voraussetzung ebenfalls das Produkt!

4) Für den Datenteil eines Programmes liegen die folgenden Erklärungen vor:

```
Ø1  A.
    Ø2  B  PIC S999  VALUE +85.
    Ø2  C  PIC X(4)  VALUE 'SPQR'.
    Ø2  D  PIC XBX   VALUE 'M B'.
Ø1  E.
    Ø2  D  PIC XX    VALUE 'VW'.
    Ø2  B  PIC S999  VALUE +43      COMP-3.
    Ø2  F  PIC X(3)  VALUE '768'.
```

Welches Resultat ergibt sich für A nach Ausführung der folgenden Statements?

a) `MOVE CORRESPONDING E TO A.`
b) `MOVE CORRESPONDING A TO E;  MOVE CORRESPONDING E TO A.`

5) Gegeben sind folgende Erklärungen im Datenteil:

```
77  G  PIC S999  SIGN LEADING.
77  H  PIC S999  SIGN TRAILING.
77  I  PIC S999  SIGN LEADING SEPARATE.
77  J  PIC S999  SIGN TRAILING SEPARATE.
```

Welche Werte sind G, H, I und J nach Ausführung der folgenden Statements zugewiesen?

a) `MOVE +983 TO G, H, I, J.`
b) `MOVE -76 TO G, H, I, J.`

19.8. EA-Dateien

Dateien, die im SELECT-Satz des Maschinenteils einer SEDZ [1] zugeordnet sind, können mit den folgenden Statements angesprochen werden:

a) OPEN I-O {dateiname} ···
b) REWRITE satzname [FROM bez]
[INVALID KEY imperativanweisung]

1) SEDZ ist die Abkürzung von Speichereinheit mit direkter Zugriffsmöglichkeit.

Die Klausel I-O kann zusammen mit den bereits bekannten Klauseln INPUT und OUTPUT in einem OPEN-Statement auftreten, wobei die Reihenfolge völlig gleichgültig ist. Die Positionierungsattribute, die nach Feststellung des Endes eines Datenträgers angezogen werden, dürfen in der Klausel I-O nicht erscheinen.

Eine für die Eingabe und für die Ausgabe eröffnete Datei gilt hinsichtlich der Verarbeitung grundsätzlich zunächst einmal als Eingabedatei mit der zusätzlichen Eigenschaft, daß das o.a. REWRITE-Statement einem READ-Statement nachfolgen kann, wodurch der zuletzt gelesene Satz wieder zurückgestellt wird. Nur diese eine Form des REWRITE-Statements ist für EA-Dateien gestattet. Die Klausel INVALID KEY wird dann ausgeführt, wenn infolge falschen Schlüssels kein externer Speicherplatz zur Aufnahme des Satzes zur Verfügung steht. Fehlt diese Klausel, so ist keine spezielle Maßnahme vorgesehen, wenn die Bedingung „ungültiger Schlüssel" auftritt. Das REWRITE-Statement in dieser Form kann nur dazu benutzt werden, einen Satz durch einen anderen mit der gleichen Länge zu ersetzen.

Bei EA-Dateien besitzt das CLOSE-Statement die bereits bekannte Form; die Angabe von NO REWIND ist jedoch nicht möglich.

19.9. Definition und Verarbeitung indexsequentiell organisierter Dateien

Indexsequentiell organisierte Dateien werden mit Ausnahme der Paragraphen FILE-CONTROL und I-O-CONTROL des Maschinenteils genau so definiert wie konsekutiv organisierte Dateien.

Im Auswahlsatz (SELECT-Satz) besitzt die Zuweisungsklausel die folgende Form:

a) Betriebssystem OS

```
ASSIGN TO DA [-gerät] -I-ddname
```

b) Betriebssystem DOS

```
ASSIGN TO SYSnnn-DA-gerät-I [-name]
```

Bei indexsequentiell organisierten Daten ist außerdem eine Zugriffsklausel zu formulieren. Sie lautet:

```
ACCESS MODE IS { SEQUENTIAL }
               { RANDOM     }
```

Bei Dateien, zu deren Sätzen sequentiell zugegriffen werden soll, kann diese Klausel entfallen, d.h. standardmäßig gilt

```
ACCESS MODE IS SEQUENTIAL
```

Wird die RESERVE-Klausel benutzt, so darf in ihr nur der Vermerk NO erscheinen, wenn der direkte Zugriff vorgesehen ist.

Für jede indexsequentiell organisierte Datei ist die Aufnahme der Klausel

```
RECORD KEY IS datenname
```

in den Auswahlsatz erforderlich. Hierbei ist datenname der Name eines Feldes, das zwischen 1 und 255 Bytes lang sein kann und in die Satzbeschreibung im Dateienkapitel eingeschlossen ist. Mit diesem Feld sind die Schlüssel ansprechbar. Gehört zur Datei mehr als eine Satzbeschreibung, so muß ein äquivalentes Feld (nicht notwendig mit dem

gleichen Namen) in jeder Satzbeschreibung enthalten sein. COBOL unterstützt nämlich indexsequentiell organisierte Dateien nicht, in deren Sätzen keine eingebetteten Schlüssel vorhanden sind.

Bei vorgesehenem direkten Zugriff muß der Auswahlsatz auch um die Klausel

NOMINAL KEY IS datenname

bereichert sein. Unter datenname ist hierbei ein Datenelement aus dem Arbeitsspeicherkapitel zu verstehen, das nicht in einem Satz hinter einem Datenelement definiert sein darf, das mit der Klausel

OCCURS DEPENDING ON ···

ausgezeichnet ist. Diesem Datenelement muß als Wert der Schlüssel desjenigen Satzes zugewiesen sein, der zur Eingabe kommen bzw. zur Datei hinzugefügt oder ersetzt (Ausgabe) werden soll. Bei Dateien, zu denen sequentiell zugegriffen werden soll, ist diese Klausel dann und nur dann erforderlich, wenn das START-Statement benutzt wird. Wir behandeln dieses Statement etwas später, jedoch noch in diesem Abschnitt.

Für Dateien, die der direkten Zugriffsmethode unterliegen sollen, kann noch die Klausel

TRACK-AREA IS {datenname / gz} CHARACTERS

benutzt werden. Diese Klausel ist nur dann zu verwenden, wenn die Datei um neue Sätze erweitert werden soll. Durch sie wird ein Gebiet festgelegt, das als Arbeitsgebiet benutzt wird; es darf nicht mehr Bytes als maximal 32767 umfassen. Dieses Gebiet, unter dem Namen datenname ansprechbar [1], wird bei der Eröffnung der Datei eingerichtet und bei ihrem Abschluß wieder freigegeben. Zu datenname muß entweder die Stufennummer 01 oder die Stufennummer 77 gehören. Wird unter gz eine natürliche Zahl kleiner als 32767 niedergeschrieben, so erzeugt der Kompilierer ein entsprechend großes Gebiet. Zur optimalen Leistungserhöhung eines Programmes sollte das Gebiet so groß sein, daß es einmal alle Schlüssel und alle Datensätze aufnehmen kann, die auf einer Spur gespeichert sind, und außerdem Platz frei hat für einen zusätzlichen Schlüssel und einen zusätzlichen Satz [2].

Im Paragraphen I-O-CONTROL erlaubt das Betriebssystem OS die Aufnahme der Klauseln

APPLY CORE-INDEX ON {dateiname} ···
APPLY REORG-CRITERIA TO datenname ON dateiname

Die erste dieser beiden Klauseln kann nur für Dateien benutzt werden, zu deren Sätzen direkt zugegriffen werden soll. Durch die Niederschrift von CORE-INDEX ist festgelegt, daß die höchste Indexstufe im Hauptspeicher gehalten wird, solange die Datei eröffnet ist.

---

1) Im Betriebssystem DOS ist die Angabe von datenname nicht möglich. Außerdem ist für gz die Bedingung

gz - 24 + n (40 + 5 + b)

einzuhalten. Hierbei bedeuten:

n | Anzahl der Blöcke auf der Spur
s | Schlüssellänge
b | Blocklänge

Wenn n gleich 1 ist, so ist die Zahl 40 in der Formel durch die Zahl 50 zu ersetzen.

2) Im Betriebssystem OS muß die Gebietslänge als ein Vielfaches von 8 definiert werden.

REORG-CRITERIA wird in Verbindung mit dem DD-Statement der Jobsteuersprache benutzt, um acht Informationsbytes unter datenname zu speichern, wenn die Datei abgeschlossen wird. Die Erklärung für datenname muß folgendes Aussehen haben:

```
xx  datenname COMPUTATIONAL.
  yy  FILLER PICTURE S999.
  yy  FILLER PICTURE S999.
  yy  FILLER PICTURE S9(6).
```

Das erste Halbwort nimmt die Zahl der vollen Zylinderüberlaufbereiche der Datei auf, das zweite Halbwort die Zahl der nicht voll belegten Spuren des unabhängigen Überlaufbereiches. Das letzte Wort schließlich enthält die Anzahl der Eingabe- bzw. Ausgabestatements, die zu Überlaufsätzen führten, die nicht die ersten von ihrer Primärspur waren. Diese Informationen lassen sich recht gut zur Bestimmung des Zeitpunkts einsetzen, zu dem eine Reorganisation der Datei erforderlich ist.

An Stelle der soeben besprochenen beiden APPLY-Klauseln für das Betriebssystem OS treten im Betriebssystem DOS die folgenden vier Klauseln:

```
APPLY CYL-OVERFLOW OF gz TRACKS ON {dateiname} ...
APPLY CORE-INDEX TO datenname ON {dateiname} ...
APPLY MASTER-INDEX TO gerät ON {dateiname} ...
APPLY CYL-INDEX TO gerät ON {dateiname} ...
```

CYL-OVERFLOW kann allein für die Erstellung oder für die Erweiterung einer Datei herangezogen werden; durch diese Klausel wird die Anzahl der Spuren benannt, die in jedem Zylinder für das Überlaufgebiet bereitgestellt werden sollen (gz ist als natürliche Zahl anzugeben). Fehlt diese Klausel, so werden $\frac{1}{5}$ der Spuren eines Zylinders für Überlaufzwecke reserviert. Bei vorgesehenem direkten Zugriff ist die Angabe der Klausel CORE-INDEX recht nützlich; durch sie wird während der Eröffnungszeit der Datei die Bereitstellung von Hauptspeicherplatz für das Zylinderinhaltsverzeichnis erreicht. Dieser Speicherplatz muß wenigstens $(n+3)(s+6)$ Bytes groß sein, n bedeutet hier die An zahl der Eintragungen im Zylinderindex, die im Hauptspeicher festgehalten werden sollen, s ist die Schlüssellänge. Wird mehr als ein Dateiname aufgeführt, so muß beachtet werden, daß in jedem Augenblick nur eine einzige Datei geöffnet sein darf. Wird die Klausel MASTER-INDEX notiert, so ist unter gerät die Typennummer des Gerätes zu nennen, das für die Aufnahme des Haupt- und des Zylinderinhaltsverzeichnisses vorgesehen ist. Unter gerät können die Typennummern 2311, 2314, 2321, 3330 und 3340 genannt werden. Wird CYL-INDEX geschrieben, so gibt es kein Hauptinhaltsverzeichnis, das unter gerät genannte externe Speichergerät dient dann allein zur Aufnahme des Zylinderinhaltsverzeichnisses. Wird weder MASTER-INDEX noch CYL-INDEX niedergeschrieben, so gibt es einmal kein Hauptinhaltsverzeichnis und zum anderen residiert das Zylinderinhaltsverzeichnis auf dem gleichen externen Speichergerät, auf dem der primäre Datenbereich reserviert ist.

Die Statements, die zum Ansprechen einer indexsequentiell organisierten Datei herangezogen werden dürfen, sind in der Abb. 19.8. aufgeführt.

Eröffnen von Dateien

```
OPEN   [INPUT {dateiname} ... ]
       [OUTPUT {dateiname} ...]
       [I-O {dateiname} ...   ]
```

Einstellen einer Datei mit anschließendem Lesen

DOS:

```
START dateiname  [KEY IS {EQUAL TO / =} bez]
                 INVALID KEY imperativanweisungen
```

OS:

```
START dateiname  [USING KEY datenname {EQUAL TO / =} bez]
                 [INVALID KEY imperativanweisungen]
```

Lesen von einer Datei

```
READ dateiname RECORD [INTO bez]
               {AT END / INVALID KEY} imperativanweisungen
```

Schreiben in eine Datei

```
WRITE satzname [FROM bez]
               [INVALID KEY imperativanweisungen]
```

Zurückschreiben in eine Datei

```
REWRITE satzname [FROM bez]
                 [INVALID KEY imperativanweisungen]
```

Abschließen von Dateien

```
CLOSE { dateiname [WITH LOCK / WITH DISP / WITH POSITIONING] } ...
```

Abb. 19.8. Statements zum Ansprechen indexsequentiell organisierter Dateien

Im OPEN-Statement dürfen die Klauseln in einer beliebigen Reihenfolge genannt werden. Wird im CLOSE-Statement DISP oder POSITIONING niedergeschrieben, so wird eine solche Aussage im Betriebssystem OS ignoriert. Wie schon in der Abb. 19.8. angedeutet, gilt die erste Form des START-Statements für das Betriebssystem DOS, die zweite Form für das Betriebssystem OS. Im DOS ist die Angabe der Klausel INVALID KEY in den WRITE- und REWRITE-Statements Pflicht. Im DOS ist es weiterhin verboten, im CLOSE-Statement DISP oder POSITIONING anzugeben.

Wird eine Ausgabedatei mit sequentiellem Zugriff eröffnet, so bedeutet das die Erstellung einer neuen oder die Erweiterung einer existierenden Datei. Ein solches Vorgehen ist zugleich die einzige Methode, für die Erstellung einer indexsequentiell organisierten Datei. Die Angabe NOMINAL KEY wird dabei nicht benötigt. Neben den Statements OPEN und CLOSE darf nur noch das WRITE-Statement niedergeschrieben werden. Die Bedingung „ungültiger Schlüssel" tritt immer dann auf, wenn der Schlüssel des einzuspeichernden Satzes in der Sortierfolge kleiner als oder gleich dem Schlüssel des zuvor eingespeicherten Satzes ist.

Auf eine als Eingabedatei (INPUT) mit sequentiellem Zugriff eröffneten Datei dürfen nur die Statements OPEN, START, READ und CLOSE bezogen werden. Die Klausel

NOMINAL KEY spielt dann und nur dann eine Rolle, wenn das START-Statement benutzt wird. Im START-Statement ist unter datenname der Name des im Satz liegenden Schlüssels anzugeben, so wie er im Dateienkapitel erklärt ist, unter bez ist ein Datenelement aufzuführen, das die gleiche Verwendungsklausel wie datenname besitzt und dessen Länge kleiner oder gleich der Länge des mit datenname benannten Elementes ist. Fehlt in einem Programm das START-Statement, so beginnt die Verarbeitung beim ersten Satz der Datei. Liegt dagegen ein START-Statement vor, so wird der Verarbeitungsbeginn auf den ersten Satz der Datei eingestellt, für den sein eingebetteter Schlüssel, abgeschnitten gemäß der für bez definierten Länge, gleich dem Wert von bez ist. Existiert kein solcher Satz, so kommt es zum Auftreten der Bedingung „ungültiger Schlüssel" („INVALID KEY"). Fehlt die Schlüsselklausel gänzlich im START-Statement, so beginnt die Verarbeitung beim ersten Satz, dessen Schlüssel gleich dem Wert des Suchschlüssels (NOMINAL KEY) ist. Existiert ein solcher Satz nicht, so kommt es wie üblich zum Auftreten der Bedingung „ungültiger Schlüssel". Es sei betont, daß die Klausel NOMINAL KEY dann und nur dann angegeben zu werden braucht, wenn die KEY-Klausel im START-Statement fehlt. Für eine Datei können beliebig viele START-Statements aufgeführt werden. Geht einem READ-Statement kein START-Statement voraus, so greift das READ-Statement stets zum nächstfolgenden Satz zu. Das READ-Statement kann nur mit der Klausel AT END eingesetzt werden.

Eine als EA-Datei (I-O) mit sequentiellem Zugriff und in genau derselben Weise verarbeitet wie eine Eingabedatei, d. h. als ob sie mit INPUT eröffnet wäre. Ergänzend kann bei einer solchen Datei zusätzlich das Statement REWRITE benutzt werden, um einen zuvor mit dem READ-Statement gelesenen Satz in die Datei zurückzustellen. Die Bedingung „ungültiger Schlüssel" wird dann erhoben, wenn die vorangegangene READ-Anweisung ebenfalls zum Auftreten dieser Bedingung führte.

Eine Datei, zu der direkt zugegriffen werden soll, kann nicht als Ausgabedatei (OUTPUT) eröffnet werden. Wird sie mit INPUT eröffnet, so kann allein das READ-Statement mit der Klausel INVALID KEY benutzt werden. Diese dadurch angesprochene Bedingung „ungültiger Schlüssel" tritt dann auf, wenn kein Satz existiert, der den unter Suchschlüssel (NOMINAL KEY) angegebenen Schlüssel besitzt.

Eine Datei, zu der direkt zugegriffen werden soll, kann auch als EA-Datei (I-O) eröffnet werden. Auf eine solche Datei darf das READ-Statement wie bei einer Eingabedatei bezogen werden, aber auch die Statements WRITE und REWRITE. Durch das WRITE-Statement wird ein neuer Satz in die Datei eingefügt. Zum Auftreten der Bedingung „ungültiger Schlüssel" kommt es dabei dann, wenn ein Satz eingespeichert werden soll, dessen Schlüssel gleich dem Schlüssel eines bereits existierenden Satzes ist, d. h. der Wert unter NOMINAL KEY ist gleich dem in einem existierenden Satz eingebetteten Schlüssel (RECORD KEY). Das REWRITE-Statement hat die Ersetzung eines bereits gespeicherten Satzes zur Folge. Diesem Statement muß deshalb das Lesen dieses Satzes vorausgehen (READ-Statement). Zum Auftreten der Bedingung „ungültiger Schlüssel" kommt es dann und nur dann, wenn diese Bedingung auch beim vorangegangenen Lesen aufgetreten ist.

Kommt es zum Auftreten der Bedingung „ungültiger Schlüssel", ohne daß die Klausel INVALID KEY präsent ist, dann wird die Bedingung ignoriert und die Programmablaufsteuerung geht zum unmittelbar folgenden Statement über. Ein solches Vorgehen wird allerdings vermieden, wenn in die Maßnahmekapitel des Prozedurteils ein Benutzungssatz aufgenommen wird. Über diese speziellen Kapitel werden wir im Abschnitt 19.12. zu sprechen haben.

Vielleicht der beste Weg, indexsequentiell organisierte Dateien vorzustellen, ist die Modifizierung zweier bereits bekannter Programme:

a) Erstellung einer Kundendatei

Diese Programm ist in Abb. 17.1. (Abschnitt 17.2.) dargestellt. Die Kundendatei ist nunmehr als indexsequentiell organisierte Datei aufzubauen und nicht mehr als konsekutiv organisierte.

Eine solche Neuorientierung führt selbstverständlich zu einer Programmänderung. In Abb. 19.9. sind die neuen Statements zusammengestellt, wobei die Seiten- und Zeilennummern denen der Abb. 17.1. entsprechen.

```
001070      SELECT TT ASSIGN TO DA-2314-I-TAPE, ACCESS IS SEQUENTIAL,
001075          NOMINAL KEY D1, RECORD KEY T1.
001180          LABEL RECORDS STANDARD, RECORDING MODE F.
001190 01   T.
001192      02  FILLER   PICTURE S9  COMPUTATIONAL.
001194      02  T1       PICTURE 9(4).
001196      02  FILLER   PICTURE X(154).
001370      WRITE T FROM D INVALID KEY GO TO P2.
```

Abb. 19.9. Beispiel für die Erstellung einer indexsequentiell organisierten Datei (Änderungen gegenüber Abb. 17.1.)

b) Rechnungsschreibung für Kunden

Dieses Programm ist in Abb. 18.19. (Abschnitt 18.17.) dargestellt. Der Zugriff zur Datei T erfolgt nunmehr direkt, als Suchbegriff dient der aus der Datei C stammende Schlüssel KUNDE. Gesucht wird nach dem im Datensatz eingebetteten Schlüssel KDR.

Die durch diese neue Aufgabenstellung bedingten Programmänderungen sind in Abb. 19.10. aufgeführt, wobei auch hier die Seiten- und Zeilennummern denen der Abb. 18.19. entsprechen.

```
ØØ112Ø      SELECT T ASSIGN TO DA-2314-I-T  ACCESS IS RANDOM
ØØ113Ø               NOMINAL KEY KUNDE, RECORD KEY KDR.
ØØ129Ø             LABEL RECORDS STANDARD, RECORDING MODE F.
ØØ14ØØ*
ØØ141Ø*
ØØ142Ø*
ØØ151Ø*
ØØ318Ø P1. OPEN  INPUT C, OUTPUT L, I-O T.
ØØ323Ø P2. READ C INTO BESTELLUNG,  AT END GO TO P12.
ØØ324Ø*
ØØ325Ø*
ØØ326Ø*
ØØ327Ø      READ T,  INVALID KEY GO TO P4.
ØØ329Ø*
ØØ331Ø*
ØØ332Ø      GO TO P2.
ØØ333Ø*
ØØ334Ø*
ØØ335Ø P5.
ØØ358Ø*
ØØ359Ø*
ØØ36ØØ*
ØØ4Ø1Ø*
ØØ412Ø      REWRITE EINGABE.
ØØ415Ø P12.  CLOSE  C T L.
```

Abb. 19.10. Definition und Verarbeitung von Dateien, die nach VSAM organisiert sind.

Man beachte, daß dieses Programm hinsichtlich der Anzahl der zu schreibenden Zeilen wesentlich kürzer ist als das ursprüngliche Programm von Abb. 18.19., auch ist es augenscheinlich leichter zu verfolgen. (Überflüssige Zeilen sind in Abb. 18.19. durch ein Sternzeichen auf Spalte 7 gekennzeichnet, sind also zu einem nichtssagenden Kommentar herabgestempelt.) Der Hauptspeicherbedarf für das neu abgefaßte Programm ist jedoch höher als der des ursprünglichen Programmes, was durch den größeren Aufwand für die indexsequentielle Organisationsform und die direkte Zugriffsmethode bedingt ist.

19.10. Definition und Verarbeitung von Dateien, die nach VSAM organisiert sind

Nach VSAM organisierte Dateien sind im großen und ganzen genau so zu definieren wie konsekutiv organisierte; allerdings sind im Paragraphen FILE-CONTROL des Maschinenteils gewisse Sonderregelungen zu beachten.

Für die beiden verschiedenen Varianten von Datenmengen, die der Organisationsform VSAM unterliegen, besitzen die Auswahlsätze (SELECT-Sätze) die in Abb. 19.11. dargestellte Gestalt.

Datenmengen in	
Zugangsfolge (ESDS)	Schlüsselfolge (KSDS)
SELECT [OPTIONAL] dateiname ASSIGN TO sysname ··· [RESERVE angabe] [ACCESS MODE IS SEQUENTIAL] [ORGANIZATION IS SEQUENTIAL] [FILE STATUS IS datenname].	SELECT dateiname ASSIGN TO sysname ··· [RESERVE angabe] [ACCESS MODE IS { SEQUENTIAL / RANDOM / DYNAMIC }] ORGANIZATION IS INDEXED [FILE STATUS IS datenname] RECORD KEY IS datenname.

Bemerkung: ESDS — Entry Sequenced Data Sets
KSDS — Key Sequenced Data Sets

Abb. 19.11. SELECT-Sätze für Dateien die der Organisationsform VSAM unterliegen

Wenden wir uns zunächst den Dateien in Zugangsfolge zu. Das reservierte Wort OPTIONAL muß dann codiert werden, wenn bei der Programmausführung die betreffende Datei übergangen werden kann. Unter sysname ist ein mehrfacher Name mit folgendem Aussehen zu verstehen:

a) [DA] [-gerät] -AS-ddname *im OS*

b) SYSnnn [-DA] [-gerät] -AS [-name] *im DOS*

Werden DA und gerät niedergeschrieben, so werden beide Angaben ignoriert. Für die RESERVE-Klausel gilt das gleiche. Standardmäßig werden die Angaben über die Zugriffsmethode (ACCESS-Klausel) und Organisationsform (ORGANIZATION) in den in Abb. 19.11. aufgeführten Formen unterstützt, so daß sie auch weggelassen werden dürfen. In der Klausel FILE STATUS ist auf ein Datenelement namens datenname im Arbeitsspeicherkapitel Bezug zu nehmen, das die Abbildung XX besitzt. Nach jeder über die Datei laufenden Operation werden die beiden Zustandsbytes (Statusbytes) so gesetzt, wie es die Abb. 19.12. zeigt.

Rückmeldecode	*Bedeutung*
00	Erfolgreich ausgeführte Operation
10	Dateiende (EOF)
21	Reihenfolgefehler
22	Doppelter Schlüssel
23	Kein Satz gefunden
30	Physischer EA-Fehler
34	Verletzung der Grenzwerte (für Dateien in Zugangfolge)
92	Logischer Fehler
93	Externes Speichergerät nicht verfügbar
94	Kein Satzzeiger für sequentielle Anforderung vorhanden
95	Unvollständige oder ungültige Dateiinformationen

Anmerkung: EOF ist die Abkürzung von End of File

Abb.19.12. Wert und Bedeutung der Zustandsbytes bei Dateien in Zugangsfolge (Rückmeldecodes)

Gehen wir nun zur Besprechung der Dateien in Schlüsselfolge über. Unter sysname kann mit einer Ausnahme die gleiche Zusammenstellung erfolgen wie bei Dateien in Zugangsfolge. AS darf als Hinweis auf die Organisation jedoch nicht gebracht werden. In der RECORD-Klausel kann auf ein alphamerisches Datenelement fester Länge bezuggenommen werden, das innerhalb einer der für die Datei definierten Sätze liegen muß.

Fehlt die Angabe der Zugriffsklausel, so wird die sequentielle Zugriffsmethode unterstellt. Bei der sequentiellen Verarbeitung werden die Sätze der Reihenfolge zur Verfügung gestellt, die durch die in die Sätze eingebetteten Schlüssel (Satzschlüssel) gegeben ist. Bei der direkten Verarbeitung muß das Programm im Schlüsselfeld des Satzes den Schlüssel des zu lesenden oder zu schreibenden Satzes finden. Die Angabe DYNAMIC erlaubt sowohl die sequentielle Verarbeitung als auch die Direktverarbeitung der Datei. Nach jeder Operation wird ein Rückmeldecode in das Feld gestellt, das unter datenname bei der STATUS-Klausel angegeben ist. Die Rückmeldecodes sind mit einer Ausnahme die gleiche, wie sie in der Abb. 19.12. aufgeführt sind. Der für Dateien in Zugangsfolge vorhandene Rückmeldecode 34 ist nicht definiert, der gleiche auch hier mögliche Fehler wird mit der Zeichenkombination 24 angezeigt.

Hinsichtlich der für die Verarbeitung von Dateien, die virtuell sequentiell organisiert sind, haben wir einige Aussagen zu machen.

Das OPEN-Statement nimmt bezüglich seiner Dateiverwendungsklauseln (INPUT, OUTPUT, I-O) die bereits bekannten Formen an. Hinzu kommen kann der Verwendungshinweis EXTEND. OUTPUT kann dann und nur dann geschrieben werden, wenn die Datei angelegt werden soll, d.h. zuvor noch keine Sätze in die Datei geschrieben wurden, anderenfalls ist EXTEND zu benutzen. EXTEND verursacht die Hinzufügung von Sätzen am Dateiende. Welche EA-Statements für eine Datei hinausgeschickt werden dürfen, hängt ausschließlich von den Dateiverwendungshinweisen im OPEN-Statement ab (Abb. 19.13.).

Dateiverwendungs-hinweis	Zuverlässige EA-Statements
INPUT	START READ
OUTPUT	WRITE
EXTEND	WRITE
I-O	START READ WRITE REWRITE

Abb. 19.13. Zulässige EA-Statements

Das START-Statement sieht grundsätzlich genau so aus wie das gleiche Statement bei indexsequentiell organisierten Dateien (siehe Abschnitt 19.9.). Es dürfen jedoch weit mehr Operatoren aufgenommen werden, nämlich insgesamt folgende:

```
EQUAL TO
=
GREATER THAN
>
NOT LESS THAN
NOT <
```

Fehlt das START-Statement, so beginnt die sequentielle Verarbeitung am Anfang der Datei. Unter der Bezeichnung bez im START-Statement ist der Name eines alphamerischen Datenelementes niederzuschreiben, dessen linksbündiges Byte beim Anfang des Satzschlüssels beginnt. Das Datenelement selbst kann kürzer als der Satzschlüssel sein. Fehlt die KEY-Klausel, so wird standardmäßig

```
KEY = sschl
```

unterstellt, wobei sschl der Satzschlüssel, d.h. der im Satz eingebettete Schlüssel ist. Die Benutzung der Klausel INVALID KEY ist für virtuell sequentiell organisierte Dateien freigestellt.

In welcher Form das READ-Statement zu benutzen ist, hängt von der Zugriffsmethode ab. Es gilt für die

a) sequentielle Zugriffsmethode

```
READ dateiname [NEXT] RECORD [INTO bez]
               [AT END imperativanweisungen]
```

b) direkte Zugriffsmethode

```
READ dateiname RECORD [INTO bez]
               [INVALID KEY imperativanweisungen]
```

Für Dateien, bei denen für den Zugriff DYNAMIC erklärt ist (Abb. 19.11.), ist das Wort NEXT erforderlich, um einen sequentiellen und nicht einen direkten Zugriff anzuzeigen.

Zur Hinzufügung eines neuen Satzes dient das Statement

WRITE satzname [FROM bez] [INVALID KEY imperativanweisungen]

Für Dateien, in die die Sätze sequentiell aufgezeichnet werden sollen, ist die Klausel INVALID KEY nicht anwendbar. Außerdem müssen hierbei die Schlüssel der einzufügenden Sätze in aufsteigender Reihenfolge vorliegen.

Um einen bereits eingespeicherten Satz zu ersetzen, ist das Statement

REWRITE satzname [FROM bez] [INVALID KEY imperativanweisungen]

zu benutzen. Auch für dieses Statement gilt, daß die Klausel INVALID KEY bei Dateien, zu denen sequentiell zugegriffen werden soll, nicht erlaubt ist. Dem REWRITE-Statement muß außerdem ein erfolgreich ausgeführtes READ-Statement vorangegangen sein.

Für EA-Dateien (I-O im OPEN-Statement) ist ferner die Entfernung gespeicherter Sätze durch die Verwendung des Statements

DELETE dateiname RECORD [INVALID KEY imperativanweisungen]

möglich. Bei vorgesehenem sequentiellen Zugriff muß die Klausel INVALID KEY weggelassen werden. Dem DELETE-Statement muß außerdem eine beanstandungsfreie Leseoperation vorausgegangen sein. Bei der Direktverarbeitung wird der Satz entfernt, der durch den Satzschlüssel bestimmt ist.

Schließlich besitzt das CLOSE-Statement die folgende Form:

CLOSE dateiname [WITH LOCK] ...

Durch die Angabe von LOCK wird verhindert, daß die Datei im gleichen Programm später noch einmal eröffnet wird.

19.11. Definition und Verarbeitung von Dateien, die der gestreuten (direkten) Organisationsform unterliegen

Bei einer Datei, die der gestreuten (direkten) Organisationsform unterliegt, muß der COBOL-Programmierer die Spur auf der SEDZ angeben, auf der der Satz liegt und außerdem den Schlüssel des gesuchten Satzes. Zunächst erfolgt eine Einstellung des Zugriffsmechanismus auf die angegebene Spur. Wenn sich der Satz dort befindet, so wird er in den Hauptspeicher eingelesen Im anderen Fall wird das Durchsuchen mit der nächsten Spur fortgesetzt. Das Durchsuchen wird in dem Augenblick abgeschlossen, in dem entweder der Satz gefunden oder das Dateiende festgestellt wurde. Es ist dem Programmierer freigestellt, das Durchsuchen auf eine vorgegebene Anzahl von Spuren zu begrenzen. Daneben ist es möglich, Dateien mit gestreuter Organisationsform aufzubauen, die das Ansprechen ihrer Sätze mittels relativer Satznummern erlauben. Für diese Dateien wird in COBOL oft der Begriff Relativdateien gebraucht. Wir wollen uns dieser Terminologie anschließen.

Wenden wir uns zunächst der Definition derjenigen Dateien zu, deren Sätze mittels Spurnummer und Schlüssel angesprochen werden.

Im Auswahlsatz besitzt der Systemname [1] in der Zuweisungsklausel folgende Form:

a) DA[-gerät]-D-name
   DA[-gerät]-W-name
   } *im Betriebssystem OS*

b) SYSnnn-DA-gerät-D[-name]
   SYSnnn-DA-gerät-W[-name]
   } *im Betriebssystem DOS*

Die Angabe von W im Systemnamen gestattet die Benutzung des REWRITE-Statements für EA-Dateien, die Angabe von D läßt eine solche Möglichkeit nicht zu.

Die RESERVE-Klausel darf nicht codiert werden. In der Zugriffsklausel kann sowohl auf die sequentielle Zugriffsmethode (SEQUENTIAL) oder auf die direkte Zugriffsmethode (DIRECT) Bezug genommen werden.

Mit Ausnahme des sequentiellen Lesens der Sätze einer Datei muß im Auswahlsatz die Klausel für den Suchschlüssel

```
ACTUAL KEY IS datenname
```

aufgenommen werden. Unter datenname ist dabei eine Gruppe aus zwei Datenelementen anzusprechen. Hierfür lautet die Erklärung:

```
xx  datenname.
    yy  relspur  PICTURE S9(5) COMPUTATIONAL.
    yy  schlüss  PICTURE abb.
```

Unter xx und yy sind Stufennummern zu verstehen, wobei yy höherwertiger sein muß als xx. Das erste Datenelement relspur dient zur Aufnahme der relativen Spurnummer, während das zweite, allgemein schlüss genannt, den Schlüssel aufnimmt, der auf der SEDZ aufgezeichnet werden soll bzw. worden ist. Zu diesem gehört die Abbildung abb.

Im Betriebssystem OS ist es möglich, in den Auswahlsatz (SELECT-Satz) auch die Grenzspurklausel

```
TRACK-LIMIT IS gz [TRACK ]
                  [TRACKS]
```

aufzunehmen, wobei unter gz eine nichtnegative ganze Zahl zu verstehen ist. Dabei verweist gz auf die relative Spurnummer der letzten zur Datei gehörenden Spur. Fehlt diese Klausel, so benutzt das OS den gesamten externen Speicherraum, der im DD-Steuerstatement der Jobsteuersprache für die betreffende Datei vorgesehen ist. Das gilt selbstverständlich nur für Dateien, die mittels der direkten Zugriffsmethode erzeugt werden. Werden Dateien mit der sequentiellen Zugriffsmethode generiert, so gelten nur die Spuren als belegt, die im Zeitpunkt der Ausführung des CLOSE-Statements mit Sätzen angefüllt sind. Im Betriebssystem DOS wird stets vollständig der Platz in Anspruch genommen, der in den entsprechenden Steuerstatements der Jobsteuersprache (EXTENT-Statements) festgelegt ist.

Im Betriebssystem DOS ist es erlaubt, in den Paragraphen I-O-CONTROL die Klausel

```
APPLY EXTENDED-SEARCH ON dateiname ...
```

aufzunehmen, die freilich nur für die direkte Zugriffsmethode bei den Dateiverwendungen INPUT und I-O angebbar ist. Sie bewirkt die Durchführung des erweiterten Suchens. Im Betriebssystem wird bekanntlich das erweiterte Suchen im DD-Statement der Jobsteuersprache festgelegt. Beim erweiterten Suchen kommt es zum Auftreten der Bedingung „ungültiger Schlüssel" dann und nur dann, wenn der zu suchende Satz auf keiner der zu durchsuchenden Spuren gefunden wird.

---

1) Siehe auch Abb. 19.11. Hier ist der Systemname abgekürzt mit sysname bezeichnet.

Für die Verarbeitung gestreut organisierter Dateien, deren Sätze über die relative Spurnummer und über den aufgezeichneten Schlüssel angesprochen werden, gelten die in Abb. 19.14. aufgeführten Statements.

```
OPEN [INPUT  dateiname  ...]
     [OUTPUT  dateiname  ...]
     [I-O  dateiname  ...]

SEEK dateiname RECORD

READ dateiname [INTO bez]
     AT END
     INVALID KEY          imperativanweisungen

WRITE satzname [FROM bez]
      [INVALID KEY imperativanweisungen]

REWRITE satzname [FROM bez]
        [INVALID KEY imperativanweisungen]

CLOSE [UNIT] dateiname [WITH LOCK] ...
```

Abb. 19.14. EA-Statements für gestreut (direkt) organisierte Dateien mit der Satzansprache über relative Spurnummer und aufgezeichneten Schlüssel

Im OPEN-Statement dürfen die Klauseln in beliebiger Reihenfolge aufgeführt werden. Im CLOSE-Statement darf UNIT nur dann angegeben werden, wenn zu der betreffenden Datei sequentiell zugegriffen wird. Das SEEK-Statement hingegen kann nur auf Dateien bezogen werden, zu deren Sätzen direkt zugegriffen wird.

Bevor wir weitergehen, halten wir es für angebracht, einige spezielle Ergänzungen, die die beiden Betreibssysteme OS und DOS angehen, zu bringen, Im DOS ist die Angabe der Klausel INVALID KEY bei allen WRITE- und REWRITE-Statements unbedingt erforderlich. Im OS werden alle SEEK-Statements als Kommentare betrachtet. Im OS ist es außerdem erlaubt, im CLOSE-Statement die Angaben

```
WITH DISP
```

und

```
WITH POSITIONING
```

aufzuführen; beide werden bei der Übersetzung ignoriert.

Ein paar Worte mögen zur Erklärung des Suchstatements herhalten. Die Hereinnahme des Statements SEEK in ein Programm bewirkt, daß der Zugriffsmechanismus auf die Spur eingestellt wird [1], die durch den ersten Teil des Suchschlüssels (ACTUAL KEY) bestimmt ist.

---

1) Bei Speichergeräten mit direkter Zugriffsmöglichkeit, die mit fest eingestellten Zugriffsmechanismen ausgerüstet sind, ist die Anwendung dieses Statements nutzlos.

Bei einer ungültigen Spurnummer unterbleibt die Einstellung. Beim Mehrprogrammbetrieb kann es vorkommen, daß durch dieses Statement der Leistungsgrad vermindert wird.

Wird eine gestreut organisierte Datei als Ausgabedatei für die direkte Zugriffsmethode eröffnet, so ist die Hinzufügung neuer Sätze zur Datei durch das Statement WRITE abzudecken. Die Bedingung „ungültiger Schlüssel" tritt dann auf, wenn die relative Spurnummer außerhalb der für die Datei definierten Grenzen liegt. [1)]

Die gesamte Datei wird zum Zeitpunkt der erstemaligen Eröffnung eingerichtet, d.h. der Platz für die Aufnahme der Sätze eingeteilt. Die Bedingung „ungültiger Schlüssel" kommt nicht zustande, wenn auf einer Spur nicht mehr genügend Platz vorhanden ist, um einen weiteren Satz aufnehmen zu können. In diesem Fall wird der Job abgebrochen, es sei denn, daß ein Benutzungsstatement (USE-Statement) vorliegt. Über dieses sprechen wir noch in diesem Kapitel.

Wird eine gestreut organisierte Datei als Eingabedatei für die direkte Zugriffsmethode eröffnet, so ist das READ-Statement mit der Klausel INVALID KEY zu codieren. Die Bedingung „ungültiger Schlüssel" tritt dann auf, wenn der Suchschlüssel (ACTUAL KEY) zu keinem in der Datei eingespeicherten Satz gehört. Enthält eine Spur mehr als einen Satz mit dem gesuchten Schlüssel, so gibt es keine Möglichkeit, den Satz vorauszusagen, der tatsächlich gelesen wird.

Bei einer gestreut organisierten Datei, die als EA-Datei für die direkte Zugriffsmethode eröffnet wird, kann das READ-Statement wie bei einer Eingabedatei verwendet werden. Hinzu kommt dann noch das WRITE-Statement. Folgt ein solches auf ein READ-Statement mit dem gleichen Schlüssel, so wird der gelesene (und noch gespeicherte) Satz durch denjenigen ersetzt, auf den das WRITE-Statement zugeschnitten ist. Im anderen Fall wird ein neuer Satz zur Datei hinzugefügt, genau so wie bei einer Ausgabedatei. - Voraussetzung für eine derartige Dateibehandlung ist die Angabe von D im Systemnamen in der Zuweisungsklausel (ASSIGN-Klausel) des Auswahlsatzes (SELECT-Satzes).

Bei gestreut organisierten Dateien, die im Systemnamen W und nicht D aufweisen und die als EA-Dateien für die direkte Zugriffsmethode eröffnet sind, kann das READ-Statement wie bei einer Eingabedatei und das WRITE-Statement wie bei einer Ausgabedatei verwendet werden. Zusätzlich kann noch das REWRITE-Statement (diesem muß nicht unbedingt ein READ-Statement vorausgehen) eingesetzt werden. Dieses sucht nach dem Satz und ersetzt ihn, der durch den Suchschlüssel angesprochen ist. Zum Auftreten der Bedingung „ungültiger Schlüssel" kommt es bei der Ausführung dieses Statements immer dann, wenn entweder der zu ersetzende Satz nicht gefunden werden kann oder wenn die relative Spurnummer außerhalb der für die Datei definierten Grenzen liegt. Bei gestreut organisierten Dateien, die nicht mit I-O eröffnet sind, wird das W im Systemnamen als D betrachtet.

Bei gestreut organisierten Dateien, die als Eingabedateien für die sequentielle Zugriffsmethode eröffnet sind, ist das READ-Statement mit der Klausel AT END einzusetzen. Eine so definierte und angesprochene Datei wird wie eine konsekutiv organisierte Datei behandelt. Ist dabei die Klausel ACTUAL KEY spezifiziert, so wird der Schlüssel des gerade gelesenen Satzes in die hierfür festgelegte Datengruppe hineingestellt.

Im Betriebssystem OS werden bei Vorliegen von Sätzen fester Länge die Scheinsätze ebenfalls im Programm zur Verarbeitung zur Verfügung gestellt; es sollte deshalb jeder gelesene Satz dahingehend geprüft werden, um Scheinsätze übergehen zu können.

---

1) Im Betriebssystem OS tritt diese Bedingung auch dann auf, wenn das erste Zeichen eines Satzes fester Länge der figurativen Konstanten HIGH-VALUE gleichkommt.

Im Betriebssystem OS, jedoch nicht im Betriebssystem DOS, ist der Einsatz der sequentiellen Zugriffsmethode für Ausgabedateien erlaubt. Das WRITE-Statement ist zum Schreiben der Sätze heranzuziehen. Zum Auftreten der Bedingung „ungültiger Schlüssel" kommt es hierbei, wenn die Datei vollständig gefüllt ist.

Im Betriebssystem DOS kann außerdem eine direkte Spuradressierung bei gestreut organisierten Dateien vorgenommen werden, d.h. an Stelle der relativen Spurnummer ist die tatsächliche Spuradresse als (echter) Schlüssel zu benutzen. Diese Einrichtung ist in erster Linie vorgesehen, um die Verträglichkeit mit älteren COBOL-Kompilierern zu gewährleisten. Sie gilt heute, das kann man wohl ohne Übertreibung sagen, als veraltet.

Um die tatsächliche Spuradressierung zu bekommen, ist zunächst einmal im Systemnamen (ASSIGN-Klausel im SELECT-Satz) an Stelle von D der Buchstabe A und an Stelle von W der Buchstabe U zu schreiben. Der echte Schlüssel mit dem Namen datenname ist wie folgt zu definieren:

```
xx  datenname.
    yy  echtspur  PICTURE X(8).
    yy  schlüss   PICTURE abb.
```

Es bedeuten:

xx und yy	Stufennummern (mit xx<yy)
echtspur	tatsächliche Spuradresse in der üblichen acht Bytes langen Form MBBZZKKS (Modul-Behälter-Zylinder-Kopf-Satz)
schlüss	Aufzeichnungsschlüssel, d.h. in die Spur einzutragender bzw. eingetragener Schlüssel
abb	Abbildung des Aufzeichnungsschlüssels

Bei der Codierung ist hier zunächst abseits von echtspur der Wert für echtspur zu bilden, der Wert also, der an Stelle der relativen Spurnummer tritt. Programmiertechnisch ergeben sich gewisse Schwierigkeiten. Da es z. B. keine Möglichkeit gibt, ein ein Byte umfassendes Feld mit einem Wertevorrat zu definieren, der sich aus Dualzahlen entsprechender Stellenzahl zusammensetzt, muß der Programmierer entweder ein umfassendes Feld festlegen, aus dem er dann das rechtsbündige Byte extrahiert, oder er definiert den entsprechenden Teil des Suchschlüssels neu, indem er ihn einem umfassenderen Feld überlagert. Werden M und S von echtspur während des Programmablaufes nicht geändert, so können sie mit X(1) definiert werden. Als Anfangswert ist ihnen dann die figurative Konstante LOW-VALUE zuzuweisen.

Obwohl die sogenannten Relativdateien nur im Betriebssystem OS gebildet werden können, kommt ihnen für die praktische Tätigkeit eine so große Bedeutung zu, daß es sich lohnt, sie etwas eingehender zu behandeln. Erinnern wir uns: Dateien, denen die gestreute (direkte) Organisationsform zugrundeliegt, bei denen die Sätze jedoch nicht durch die relativen Spurnummern, sondern durch ihre relativen Satznummern angesprochen werden, wollten wir, um einen Kurzausdruck einzuführen, ab diesem Abschnitt bis zum Ende des Kapitels 19 als Relativdateien bezeichnen.

Bei der Festlegung einer Relativdatei, die nur für ungeblockte Sätze fester Satzlänge (Satzformat F) gebildet werden kann, muß der Systemname in der Zuweisungsklausel des Auswahlsatzes wie folgt zusammengesetzt sein:

```
DA [-gerät]-R-name
```

Die Klausel RESERVE kann nicht angegeben werden. Als Zugriffsmethode (ACCESS) kann die sequentielle (SEQUENTIAL) oder die direkte angegeben werden. Ist der direkte Zugriff zu den Datensätzen einer Datei vorgesehen, so muß die Klausel

NOMINAL KEY IS datenname

niedergeschrieben werden, für den sequentiellen Zugriff ist die Angabe dieser Klausel freigestellt. Unter datenname ist ein Datenelement des Arbeitsspeicherkapitels mit der Abbildung S9(8) und mit der Angabe COMPUTATIONAL zu verstehen. Dieses Datenelement dient zur Aufnahme der relativen Satznummer. Liegt es innerhalb eines Satzes, so darf es nicht einem Element folgen, das die Klausel

OCCURS ··· DEPENDING ON ···

besitzt.

Mit Ausnahme des SEEK-Statements können alle anderen EA-Statements benutzt werden (siehe Abb. 19.14.). Bei Benutzung der direkten Zugriffsmethode können Dateien nicht als Ausgabedateien (OUTPUT) eröffnet werden.

Relativdateien werden mittels der sequentiellen Zugriffsmethode erstellt, wobei sie als Ausgabedateien (OUTPUT) eröffnet werden. Durch das WRITE-Statement wird ein Satz in die nächste freie Region hineingestellt. Ist der Schlüssel (NOMINAL KEY) angegeben, so wird der Satz in der Region untergebracht, die durch die spezifizierte relative Satznummer ausgezeichnet ist; in dazwischen liegende unbenutzte Regionen werden Scheinsätze geschrieben. Scheinsätze besitzen auf ihrem ersten Byte den Wert der figurativen Konstanten HIGH-VALUE. Bei Fehlen des Schlüsselfeldes können Scheinsätze per Programm eingefügt werden. Wird die Klausel NOMINAL KEY jedoch benutzt, so wird die relative Satznummer des letzten aufgezeichneten Satzes in das Schlüsselfeld hineingestellt, und zwar geschieht das nach jeder Operation, die durch ein Statement WRITE, CLOSE UNIT oder CLOSE veranlaßt wurde. Durch das CLOSE-Statement wird das Füllen der restlichen Regionen auf der letzten Spur mit Scheinsätzen durchgeführt, durch CLOSE UNIT sogar das Füllen der Regionen aller noch verbleibender zur Datei gehörender Spuren mit Scheinsätzen. Die Bedingung „ungültiger Schlüssel" tritt dann und nur dann auf, wenn der angegebene Schlüssel außerhalb der für die Datei vorgesehenen Grenzen liegt.

Bei Relativdateien, die als Eingabedateien mit sequentiellem Zugriff eröffnet werden, ist das READ-Statement mit der Dateiendeklausel (AT END) einzusetzen. Wird die Klausel NOMINAL KEY benutzt, so wird nach jedem Lesen die relative Satznummer in das Schlüsselfeld hineingestellt. Da Scheinsätze dem Programm ebenfalls zur Verfügung gestellt werden, sind entsprechende Abfragevorkehrungen zu treffen.

Bei Relativdateien, die als Eingabedateien für den direkten Zugriff vorgesehen sind, ist das READ-Statement mit der Klausel INVALID KEY zu benutzen. Zum Auftreten dieser Bedingung kommt es immer dann, wenn ein Leseversuch mit einem ungültigen Schlüssel unternommen wird, d.h. mit einer außerhalb des Dateiumfanges liegenden relativen Satznummer.

Bei vorgesehenem direkten Zugriff auf EA-Dateien können die Statements READ und REWRITE codiert werden. Dem letzteren Statement muß stets ein READ-Statement vorangehen. Durch das REWRITE-Statement wird dann der gerade gelesene Satz ersetzt. Für beide Statements, READ wie auch REWRITE, kommt es zum Auftreten der Bedingung „ungültiger Schlüssel" stets dann, wenn die unter NOMINAL KEY spezifizierte relative Satznummer außerhalb der Dateigrenzen liegt.

19.12. Die Maßnahmekapitel im Prozedurteil

Zu Beginn des Prozedurteils können spezielle Kapitel mit Maßnahmen niedergeschrieben werden. Die in diesen Kapiteln enthaltenen Statements werden niemals im Rahmen der normalen Ablauffolge eines Programmes ausgeführt. Zu ihrer Ausführung kommt es nur dann, wenn eine bestimmte anormale Bedingung während des Programmablaufes eingetreten ist. Gewöhnlich werden derartige Maßnahmen für die Behandlung von

- EA-Fehlern
- ungültigen Schlüsseln
- nichtstandardisierten Kennsätzen bei Magnetbändern

usw. vorgesehen. Alle Maßnahmekapitel werden zusammengefaßt durch die Vereinbarung

DECALRATIVES.

eingeleitet und durch die Vereinbarung

END DECLARATIVES.

abgeschlossen. Ihre allgemeine Form ist in Abb. 19.15. dargestellt.

```
        .
        .
        .
PROCEDURE DIVISION.
DECLARATIVES.
{kapitelname SECTION.  USE satz.
{paragraphenname.  {stmts} ...} ...} ...
END DECLARATIVES.
        .
        .
        .
```

Abb. 19.15. Einleitung des Prozedurteils mit Maßnahmenkapiteln

Die beiden Randvereinbarungen DECLARATIVES und END DECLARATIVES müssen innerhalb der Randbegrenzung A anfangen. Der restliche Teil der Codierung muß den üblichen Regeln genügen. Jede Maßnahme stellt ein gesondertes Kapitel dar. Der erste Satz jedes Kapitels ist ein Benutzungssatz (USE-Satz). Er wird niemals ausgeführt; er besagt nur dem Kompilierer, unter welcher Bedingung die im Kapitel festgelegte Maßnahme angezogen werden soll.

Innerhalb des Hauptprogrammes können nur sehr beschränkt Bezugnahmen auf die codierten Maßnahmen vorgenommen werden. Die einzige Ausnahme bildet das PERFORM-Statement, das die Ausführung einer Prozedur (Kapitel oder Paragraph) aus den Maßnahmen veranlassen kann. Innerhalb der Maßnahmen ist jedoch die Benutzung von Statements des Hauptprogrammes untersagt.

Aus einem Maßnahmekapitel erfolgt der Ausgang in folgender Weise:

a) Nach Ausführung des letzten Statements des betreffenden Kapitels erfolgt die normale Rückkehr zu dem Statement, das die Ausführung der Maßnahme hervorrief.

b) Als anormaler Ausgang kann ein unbedingtes Verzweigungsstatement (GO TO) codiert werden. Allerdings geht dabei der normale Rückkehrpunkt des Hauptprogrammes verloren.

Wir wollen unsere Aufmerksamkeit hauptsächlich auf solche Maßnahmen lenken, die bei der Behandlung von EA-Fehlern und von ungültigen Schlüsseln von Nutzen sind. Hierbei weist der Benutzungssatz die folgende Gestalt auf:

```
USE AFTER STANDARD ERROR PROCEDURE ON
    { dateiname-1  dateiname-2  ... }
    { INPUT                        }
    { OUTPUT                       } [GIVING datenname-1 [datenname-2]].
    { I-O                          }
    { EXTEND                       }
```

Hinsichtlich der Betriebssysteme gelten folgende Sonderregeln:

1) Die 2. Version des OS-Kompilierers gestattet nicht den Einsatz der GIVING-Klausel, wenn INPUT, OUTPUT, I-O oder EXTEND angegeben sind. Die Versionen 2 und 3 des Kompilierers erlauben nicht die Verwendung von EXTEND.

2) Die Kompilierer des DOS erlauben die Verwendung der GIVING-Klausel nur dann, wenn ein einziger Dateiname angegeben ist. Einige Kompilierer lassen die Benutzung von EXTEND nicht zu.

Wird INPUT angegeben, so wird die Maßnahme auf alle als Eingabedateien eröffneten Dateien angewendet, bei Angabe von OUTPUT auf alle als Ausgabedateien eröffneten Dateien. In gleicher Weise treffen I-O und EXTEND auf diejenigen Dateien zu, die mit I-O bzw. EXTEND eröffnet wurden. Fehlen die Dateiverwendungsangaben, so wird die codierte Maßnahme für die mit dateiname-1, dateiname-2 usw. benannten Dateien angezogen. Es ist nicht möglich, mehr als einen Benutzungssatz für eine Datei zu kreieren.

Wird datenname-1 niedergeschrieben, so ist mit diesem ein 136 Bytes (Betriebssystem OS) bzw. 8 Bytes (Betriebssystem DOS) umfassendes Datenelement gemeint, in das vom jeweiligen Betriebssystem Informationen über die Art des aufgetretenen Fehlers hineingestellt werden. Das Datenelement muß im Arbeitsspeicherkapitel des Datenteils definiert sein. Beim Betriebssystem DOS ist es außerdem erforderlich, daß das Element als externes Dezimalelement erklärt wird. Wird datenname-2 niedergeschrieben, so nimmt das dadurch angesprochene Datenelement bei einem READ-Statement den fehlerhaften Satz auf.

In der Abb. 19.16. sind die Bedingungen zusammengestellt, die die Ausführung von Maßnahmen bewirken, die im Benutzungssatz umrissen sind. Beim Auftreten eines physischen EA-Fehlers ist die weitere Verarbeitung einer Datei gewöhnlich nicht mehr möglich; ein gezielter Abschluß durch das CLOSE-Statement ist meist der einzige sinnvolle Weg. Bei Feststellung eines ungültigen Schlüssels bestehen meist kaum irgendwelche Beschränkungen für die weitere Verarbeitung.

Wir wollen diesen Abschnitt mit der Betrachtung einiger Besonderheiten abschließen, die speziell für die beiden Betriebssysteme gelten.

1) *Betriebssystem DOS*
Bei einem anormalen Ausgang erlaubt das DOS die Fortführung in genau der gleichen Weise wie bei einem normalen Ausgang mit Ausnahme konsekutiv organisierter Dateien, bei denen READ-Statements zum Programmabbruch führen.

2) *Betriebssystem OS*
Die Ausführung von Statements, die durch die Klausel INVALID KEY aktiviert werden, hat zur Folge, daß die vom Betriebssystem eingerichteten Steuerblöcke und Rettungsgebiete[1)]

1) Im angloamerikanischen Sprachraum als „Save areas" bezeichnet.

Fall	*Programmbausteine*	*Maßnahmen bei*	
		EA-Fehlern	*ungültigen Schlüsseln*
I	a) USE-Satz mit nachfolgender Prozedur im Programm b) KEY-Klausel im Statement	Ausführung der Prozedur, die im Anschluß an den USE-Satz codiert ist	Ausführung der Statements in der KEY-Klausel
II	a) USE-Satz mit nachfolgender Prozedur im Programm b) Keine KEY-Klausel im Statement	Ausführung der Prozedur, die im Anschluß an den USE-Satz codiert ist	* Ausführung der Prozedur, die im Anschluß an den USE-Satz codiert ist
III	a) Kein USE-Satz für die Datei b) KEY-Klausel im Statement	Systemmaßnahme	Ausführung der Statements in der KEY-Klausel
IV	a) Kein USE-Satz für die Datei b) Keine KEY-Klausel bei dem Statement, das die Maßnahme verursacht; jedoch sind KEY-Klauseln bei einigen anderen Statements vorhanden, die die betreffende Datei ansprechen	Systemmaßnahme	* Ignorierung der aufgetretenen Bedingung
V	a) Kein USE-Satz für die Datei b) Keine KEY-Klausel in irgendeinem die betreffende Datei ansprechenden Statement	Übergang zum abnormalen Abschluß des Programmes (ABEND)	Übergang zum abnormalen Abschluß des Programmes (ABEND)

Anmerkungen:

1) * → Im Betriebssystem DOS nicht möglich

2) ABEND ist die Abkürzung von „abnormal end"; es handelt sich hier um einen speziellen Programmbaustein, mit dessen Hilfe Programme anormal beendet werden, wodurch die Programmablaufsteuerung an das Betriebssystem zurückgegeben wird.

Systemmaßnahmen

Betriebssystem DOS	*Betriebssystem OS*
ABEND-Programm	a) Indexsequentiell organisierte Dateien mit direktem Zugriff → Ignorierung der Bedingung b) Indexsequentiell organisierte Dateien mit sequentiellem Zugriff → ABEND-Programm c) Andere Dateien mit direktem Zugriff → ABEND-Programm d) Andere Dateien mit sequentiellem Zugriff → gemäß EROPT-Angabe im DD-Steuerstatement der Jobsteuersprache

Abb. 19.16. Auftreten von Bedingungen und die daraus resultierenden Maßnahmen

für die inkorrekte Operation nicht freigegeben werden. Dasselbe Resultat tritt bei einem anormalen Ausgang aus einer Maßnahme ein. Unter diesen Gegebenheiten kann sich der erforderliche Hauptspeicherbedarf bei der weiteren Ausführung des Programmes beträchtlich aufschaukeln. Aus diesem Grunde sollte die Ausführung einer Maßnahme mit normalem Ausgang allen anderen Möglichkeiten vorgezogen werden, soweit das nur irgendwie opportun ist.

Aufgabengruppe 52

1) Schneide das zur Lösung der Aufgabe 1 der Aufgabengruppe 49 entwickelte Programm auf eine indexsequentiell organisierte Datei zu!

2) Schneide das zur Lösung der Aufgabe 2 der Aufgabengruppe 49 entwickelte Programm auf eine indexsequentiell organisierte Datei zu!

3) Ändere das Programm, das zur Lösung der Aufgabe 3) der Aufgabengruppe 49 geschrieben wurde,so ab, daß eine logisch fortlaufend organisierte Datei bearbeitet werden kann.

4) Eine kleinere Bank hat eine IBM /370 mit vier Plattenspeicherlaufwerken des Typs IBM 2314 installiert. Ein Plattenstapel wird als Systemresidenz benutzt, auf den drei anderen sind die laufenden Girokonten gespeichert. Über 9000 Sätze müssen dabei ständig auf dem laufenden gehalten werden.

Die einzelnen Sätze enthalten folgende Daten:

Name des Kontoinhabers	72 alphamerische Zeichen
Adresse	72 alphamerische Zeichen
Kontonummer	6 Ziffern (dezimal gepackt)
Kontostand	11 Ziffern (dezimal gepackt)
Minimaler Kontostand (1. Monat)	11 Ziffern (dezimal gepackt
Minimaler Kontostand (2. Monat)	11 Ziffern (dezimal gepackt)
Minimaler Kontostand (3. Monat)	11 Ziffern (dezimal gepackt)

Außerdem ist in jedem Satz Platz für 9 noch nicht gedruckte Kontenbewegungen gelassen. Jede dieser Kontenbewegungen setzt sich aus folgenden Informationen zusammen:

Buchungsschlüssel:	1	Ziffer (gezonte Darstellung)
Betrag :	11	Ziffern (dezimal gepackt)

Es gibt für Bewegungen die nachstehenden Buchungsschlüssel:

4	Einzahlung
5	Auszahlung
6	Zinsen
7	Spesen für Dienstleistungen und andere Gebühren
8	Gutschriften
9	Überweisungen

Kontenbewegungen werden über Lochkarten eingegeben, die wie folgt abgelocht sind:

Spalte 1:	Buchungsschlüssel
Spalten 2 bis 7:	Kontonummer
Spalten 17 bis 26:	Betrag

Ist der Buchungsschlüssel gleich 1, so liegt die Neueröffnung eines Kontos vor. In diesem Fall ist neben der Kontonummer auf den Spalten 2 bis 7 noch der Name auf den Spalten 8 bis 80 abgelocht. Durch den Buchungsschlüssel 2 wird eine neue Adresse bzw. eine Adreßänderung angezeigt; die Adresse selbst ist auf den Spalten 8 bis 80 abgelocht. Beim Buchungsschlüssel 3 ist der im zugehörigen Plattensatz ausgewiesene Kontenstand durch den Inhalt der Spalten 17 bis 26 zu ersetzen.

Schreibe ein Programm, mit dessen Hilfe Karten gelesen werden können, die sich durch die Buchungsschlüssel 1, 2 und 3 auszeichnen und führe damit den Änderungsdienst beim Girokontenbestand durch. Mit diesem Programm soll auch die Datei selbst (logisch fortlaufend organisiert) erstellt werden können.

5) Schreibe ein Programm, mit dessen Hilfe Kontenbewegungen in die Sätze der Datei geschrieben werden können, die durch das Programm von Aufg. 4) erstellt wurde. Die Felder für die minimalen Kontenstände bleiben hierbei unbeachtet. Zu Beginn des Programmes ist eine Datumskarte einzulesen; das Datum ist auf den Spalten 8 bis 20 abgelocht, beispielsweise zu 13. MAI 1968. Alle Bewegungen mit den Buchungsschlüsseln 4 bis 9 sind in die vorgesehenen Felder der Kontensätze als noch nicht gedruckte Bewegungen aufzunehmen; der Kontenstand ist außerdem zu berichtigen. Enthält eine Karte auf der Spalte 1 eine Null, so sind alle noch nicht gedruckten Bewegungen des betreffenden Kontos zu drucken und anschließend zu löschen.

6) In die Datumskarte von Aufgabe 5) ist zusätzlich auf Spalte 1 eine 1, 2 oder 3 gelocht, wodurch angezeigt werden soll, ob der 1., 2. oder 3. Monat eines Quartals vorliegt. Außerdem ist der Monatstag auf den Spalten 2 und 3 wiederholt. Liegt der Monatstag nach dem 15. des Monats, dann soll, falls Kontenbewegungen vorliegen, das Feld für den entsprechenden minimalen Kontenstand entweder vom alten Saldo oder von dem sich durch die Bewegung ergebenden neuen Saldo belegt werden, je nachdem, welcher geringer ist.

7) Schreibe ein Programm, das die Datumskarte von Aufgabe 6) einliest und anschließend automatisch bei allen Sätzen den Kontenstand in das entsprechende Feld für den minimalen Kontenstand überträgt. Dieses Programm soll nur einmal in jedem Monat laufen. Die Datumskarte enthält zur eindeutigen Kennung auf den Spalten 2 und 3 Nullen.

8) Schreibe ein Programm zur Berechnung der Vierteljahreszinsen und stelle diese mit dem Buchungsschlüssel 6 in ein Feld für noch nicht gedruckte Kontenbewegungen: Der Saldo (Kontenstand) ist außerdem um den Zinsbetrag zu berichtigen. Es soll ein Zinsfuß von 6 % p.a. (6 % jährlich) zugrundegelegt werden. Die Zinsberechnung soll wie folgt durchgeführt werden: Die drei minimalen Kontenstände sind zu addieren und mit 0,005 zu multiplizieren (aus $6/(4 \cdot 3 \cdot 100)=0,06/12$).

9) Eine Materialbestandsdatei ist indexsequentiell organisiert.

Schreibe ein Programm, das Lochkarten einliest, auf denen die Teilenummern in den Spalten 1 bis 8 abgelocht sind und suche den entsprechenden Teilesatz auf, falls ein solcher vorhanden ist. Wenn für die angegebene Teilenummer kein Satz existiert, so ist eine Fehlermeldung zu drucken; im anderen Fall ist der zugehörige Teilesatz zu drucken.

Nach Voraussetzung sind die Teilesätze 80 Stellen lang.

10) Die Aufgabe 9) ist wie folgt zu ändern:

Für die Druckausgabedatei ist die APPLY-Klausel so einzurichten, daß es möglich ist, das Auftreten der Seitenüberlaufbedingung festzustellen. Kommt es soweit, so

ist das Formular auf die nächste Seite vorzuschieben und auf die erste Zeile dieser Seite die fortlaufende Seitennummer zu drucken. Danach ist das normale Drucken fortzusetzen.

11) Nehmen wir einmal an, daß wir eine Datei vorfinden, die aus n Sätzen besteht, die alle 60 Stellen lang sind und durch PICTURE X(60) beschrieben werden können. Nach Voraussetzung ist n größer als 70 und kleiner als 2000. Wir sollen die gesamte Datei drucken; auf jeder Zeile sollen dabei zwei Sätze erscheinen. Jede Seite soll außerdem 60 Zeilen aufweisen. Somit würde z.B. die erste Seite das folgende Aussehen haben:

Satz 1	Satz 31
Satz 2	Satz 32
.	.
.	.
.	.
Satz 30	Satz 60

Auf der letzten Seite soll die Zahl der zu druckenden Zeilen auf ein Minimum reduziert werden. Sind die beiden Spalten von ungleicher Länge, so soll die linke Spalte eine Zeile mehr aufweisen als die rechte Spalte.

Bei einer umfangreichen Arbeit, wie sie z.B. das Drucken eines Telefonverzeichnisses darstellt, ergeben sich bei diesem Vorgehen erhebliche Zeitersparnisse. Diese wiegen die zusätzlichen Programmieranstrengungen, die durch das zweispaltige Drukken bedingt sind, mehr als auf.

19.13. Unterprogramme

Zu jedem COBOL-Programm gehört zunächst einmal ein Hauptprogramm; hinzu können außerdem ein oder mehrere Unterprogramme kommen. Hauptprogramm und Unterprogramme heißen gemeinsam Programmbausteine, Programmmoduln oder Programmeinheiten. Als Hauptprogramm wird dasjenige Programm bezeichnet, dem die Programmablaufsteuerung vom Betriebssystem übergeben wird. Das Hauptprogramm wiederum kann dann die Ablaufsteuerung an ein Unterprogramm übertragen. Unterprogramme brauchen nicht unbedingt in COBOL geschrieben zu sein, sie können ebenso gut auch beispielsweise in der PL/1 oder AL abgefaßt sein. Von einem Unterprogramm können andere Unterprogramme aufgerufen werden. Nach Vollendung eines Unterprogrammes wird gewöhnlich die Ablaufsteuerung zurück an die Programmeinheit gegeben, die es aufgerufen hat. Ein Unterprogramm kann jedoch auch anormal beendet werden, indem nicht nur seine eigene Ausführung abgebrochen wird, sondern auch die des Hauptprogrammes. Damit erhält das Betriebssystem die Programmablaufsteuerung zurück.

Diejenige Programmeinheit, die die Programmablaufsteuerung an eine andere überträgt, heißt aufrufende Programmeinheit, kurz auch rufendes Programm. Die Programmeinheit, die die Programmablaufsteuerung übernimmt, wird aufgerufene Programmeinheit, kurz auch aufgerufenes Programm, genannt. Zum Aufruf einer Programmeinheit wird nur das Statement

$$\underline{\text{CALL}} \left\{ \begin{matrix} \underline{\text{literal}} \\ \underline{\text{bez}} \end{matrix} \right\} \quad [\underline{\text{USING}} \ \{\underline{\text{datenname}}\} \cdots]$$

Unter literal ist der Name der Programmeinheit zu nennen, die aufgerufen werden soll. Das Literal ist zur nichtnumerischen Kategorie zu zählen, als Unterprogrammname gelten nur seine ersten acht Zeichen. Besitzt die aufgerufene Programmeinheit mehr

als eine Eingangsstelle, so nennt literal den Namen der gewünschten Eingangsstelle. Beim Aufruf wird die Programmablaufsteuerung an das erste Statement des Prozedurteils übertragen, die Maßnahmen sind davon ausgeschlossen. Jeder aufgeführte Datenname hinter dem Wort USING bezeichnet ein Argument, d.h. ein der aufgerufenen Programmeinheit zur Verfügung gestelltes Element bzw. eine Gruppe. In COBOL geschriebene Unterprogramme erwarten, daß ihnen Datennamen zur Verfügung gestellt werden. Argumente, die Unterprogrammen übertragen werden, die nicht in COBOL geschrieben sind, dürfen auch Prozedur- oder Dateinamen sein. Eine Datei, deren Name als Argument übergeben wird, muß in der aufrufenden Programmeinheit eröffnet werden. Wird im CALL-Statement an Stelle von literal eine Bezeichnung (unter bez) niedergeschrieben, so muß es sich um den Namen eines alphamerischen Datenelementes handeln, das zum Zeitpunkt der Ausführung des CALL-Statements den Namen des Unterprogrammes als Wert enthält.

Innerhalb der aufgerufenen Programmeinheit muß die Kopfvereinbarung des Prozedurteils folgendes Aussehen haben:

```
PROCEDURE DIVISION [USING {datenname} ...].
```

Soll das Unterprogramm noch an anderen (sekundären) Eingangsstellen aufgerufen werden und nicht nur an der durch die oben genannte Vereinbarung beschriebenen primären, so muß jede durch die Vereinbarung

```
ENTRY literal  [USING {datenname} ...].
```

kenntlich gemacht werden. Die ersten acht Zeichen des nichtnumerischen Literals gelten hierbei als Eingangsnamen [1], der sekundären Eingangsstelle, d.h. als der Name, auf den im CALL-Statement bezuggenommen werden muß. Wird eine Programmeinheit mittels des Namens einer sekundären Eingangsstelle aufgerufen, so wird die Programmablaufsteuerung von der aufrufenden Programmeinheit an das erste Statement übertragen, das auf die ENTRY-Vereinbarung folgt. Zwischen den Parametern [2] im Kopf des Prozedurteils und denen in der ENTRY-Vereinbarung braucht keine Übereinstimmung (weder in Zahl noch Typ) zu bestehen, es sei denn, sie besitzen die gleichen Datennamen.

Jeder in den PROCEDURE- und ENTRY-Vereinbarungen aufgeführte Dateiname muß zu einer Eintragung im Verbindungskapitel des Datenteils führen. Diese Eintragung muß dem passenden Argument der aufrufenden Programmeinheit entsprechen. In der aufgerufenen Programmeinheit dürfen weniger Argumente angesprochen werden, als die aufrufende Programmeinheit bereitstellt. In diesem Fall werden die überzähligen einfach ignoriert. Umgekehrt ist das nicht möglich: Der Versuch, mehr Argumente zu benutzen als bereitgestellt werden, resultiert gewöhnlich in einem anormalen Programmabbruch.

Das Verbindungskapitel schließt sich im Datenteil an das Arbeitsspeicherkapitel an. Es besitzt folgendes Aussehen:

```
LINKAGE SECTION.
   {dateneintragung.} ...
```

---

1) Der Name der primären Eingangsstelle ist gleich dem Namen der Programmeinheit, kurz dem Programmnamen.

2) Als Parameter bezeichnet man die Elemente bzw. Gruppen in Unterprogrammen, an die beim Aufruf die Argumente treten.

Die Dateneintragungen unterscheiden sich von denen im Arbeitsspeicherkapitel in folgenden Punkten:

a) Die Klausel VALUE darf nur für Elemente mit der Stufennummer 88 benutzt werden.

b) Nicht mehr als insgesamt 255 Eintragungen mit den Stufennummern 01 und 77 sind erlaubt.

c) Im Unterprogramm wird für die Eintragungen im Verbindungskapitel kein Hauptspeicherplatz reserviert. Das ist auch nicht nötig, da die ihnen entsprechenden Argumente ja Hauptspeicherplatz in der aufrufenden Programmeinheit beanspruchen. Der Kompilierer nimmt weiterhin an, daß jede Eintragung mit den Stufennummern 01 und 77 auf einer Doppelwortgrenze beginnt. (Bekanntlich legt ja der Kompilierer den Beginn jeder Gruppe und jedes Elementes mit den Stufennummern 01 und 77, das in Dateienkapitel bzw. im Arbeitsspeicherkapitel erklärt ist, auf eine Doppelwortgrenze. Infolgedessen muß er auch für die Eintragungen im Verbindungskapitel diese Ausrichtung annehmen.)

Da Unterprogramme unabhängig, d.h. getrennt, voneinander übersetzt werden (auch unabhängig vom Hauptprogramm), gibt es keine Namenskonflikte bei internen Namen. Etwas anderes ist das bei externen gebrauchten Namen, wie Dateinamen, Programmnamen und Eingangsnamen. Die Abbildungen und die Verwendungsklausel der Argumente können vom Kompilierer nicht dahingehend überprüft werden, ob sie mit deren der Parameter übereinstimmen. Dafür Sorge zu tragen, liegt in der Verantwortung des Programmierers selbst.

Im Betriebssystem OS ist es möglich, sogar im Prozedurteil des Hauptprogrammes einen Parameter anzugeben, dessen Name wie bei den Unterprogrammen im Verbindungskapitel erklärt werden muß. Die maximale Länge dieses Parameters darf 100 Bytes nicht überschreiten. Das diesem Parameter entsprechende Argument kommt aus dem Schlüsselwort PARM des EXEC-Steuerstatements der Jobsteuersprache, das zur Ausführung des Hauptprogrammes überleitete. Damit ist der Datentyp des Parameters sofort genannt: alphamerisch.

Beim ersten Aufruf eines Unterprogrammes besitzen die mit Anfangswerten erklärten Datenelemente diejenigen Werte, die in den VALUE-Klauseln angeführt sind. Bei allen weiteren Aufrufen muß man bei solchen Datenelementen mit den Werten rechnen, die beim vorangegangenen Aufruf ihnen zugewiesen worden sind.

Die Rückkehr von einer aufgerufenen Programmeinheit kann auf dreierlei Arten geschehen:

1) paragraphenname. EXIT PROGRAM.
2) STOP RUN.
3) GOBACK.

Wird der erste Weg praktiziert, so muß das Statement

```
EXIT PROGRAM.
```

das einzige Statement im durch paragraphenname benannten Paragraphen sein. In einem Unterprogramm sind

```
EXIT PROGRAM
```

und

```
GOBACK
```

gleichwertige Statements. Beide verursachen die Rückkehr in die aufrufende Programmeinheit, und zwar zu dem auf das CALL-Statement folgende Statement in der aufrufenden Programmeinheit. Das Statement STOP RUN hingegen bewirkt nicht nur den Abschluß der Ausführung des Unterprogrammes, sondern sogar des Hauptprogrammes, so daß die Programmablaufsteuerung zurück an das Betriebssystem geht.

In einem Hauptprogramm wird das Statement EXIT PROGRAM ignoriert, das Statement STOP RUN hingegen ist bekanntlich das Statement, das den normalen Abschluß besorgt. Nun noch ein paar Bemerkungen zum Statement GOBACK, wenn es in Hauptprogrammen codiert ist. Beim Betriebssystem DOS bewirkt es einen anormalen Programmabschluß, d.h. den Abbruch des Programmes, im OS wird es als gleichwertig dem Statement STOP RUN angesehen.

19.14. Beispiel für ein Unterprogramm

Um den Einsatz von Unterprogrammen zu demonstrieren, wollen wir das in Abb. 18.19. vorgestellte Programm etwas modifizieren.

Wir wollen annehmen, daß beim Beispiel vom Abschnitt 18.17. eine Prüfung der Kundennummer gewünscht wird, um zu sehen, ob sich aus ihr eine gültige Prüfziffer nach der Modulo-10-Prüfung (nach IBM) ergibt. Die Prüfziffer wird wie folgt gebildet, betrachten wir den Vorgang beispielsweise für die Kundennummer 123752:

a) Aus der Zahl ist jede zweite Ziffer herauszuziehen, beginnend bei der vorletzten Ziffer. Wir erhalten also

   1 3 5

b) Jeder der herausgezogenen Ziffern ist mit 2 zu multiplizieren; wenn das sich ergebende Produkt größer als 9 ist, sind die beiden Ziffern des Produktes zu addieren. In unserem Fall ergibt sich also

   2 6 1

   Bei der Multiplikation der letzten herausgezogenen Ziffer, der 5, mit 2 ergibt sich nämlich $2 \cdot 5 = 10$, so daß hier addiert werden muß; 1+0 ergibt aber 1.

c) Die so erhaltenen Ziffern sind zu addieren. Wir erhalten also

   $2 + 6 + 1 = 9$

d) Die nicht aus der Zahl herausgezogenen Ziffern sind zu addieren. Wir erhalten hier

   $2 + 7 + 2 = 11$

e) Die so erhaltenen beiden Teilsummen sind zu addieren. Dabei ergibt sich in unserem Fall:

   $9 + 11 = 20$

f) Wenn die Endsumme durch 10 teilbar ist, dann genügt die Zahl der Modulo-10-Prüfung nach IBM.

Zweckmäßigerweise werden die Kundennummern nun so vergeben, daß sie diese Eigenschaft erfüllen. Wenn nun beim Schreiben oder Ablochen einer solchen Zahl einige Ziffern vertauscht oder verwechselt werden sollten, so wird die so verstümmelte Zahl in den meisten Fällen nicht die Modulo-10-Prüfung erfüllen. Bei vielen Lochertypen kann eine Zusatzeinrichtung eingebaut werden, durch die die Prüfziffer während des Ablochens

geprüft werden kann. Wenn die Nummer nicht der Bedingung der Prüfung genügt, wird die Tastatur gesperrt und dem Bediener (der Bedienerin) dieser Fehler durch ein Signal angezeigt.

Kehren wir zu unserem Programm von Abb. 18.19. zurück. Im Arbeitsspeicherkapitel schreiben wir eine spezielle Eintragung mit der Stufennummer 77 nieder:

```
77  CHECK PICTURE 9(4)
```

Im Prozedurteil fügen wir zu Beginn des Paragraphen P5 (Zeile 003330) folgende Statements hinzu:

```
MOVE KUNDE TO CHECK.
CALL 'CHKDGT'  USING CHECK.
IF  KUNDE NOT EQUAL CHECK  GO TO P7.
```

Unserem Unterprogramm wird als Argument eine vierstellige ganze Zahl übergeben. Die Einerstelle dieser Zahl wird im Unterprogramm durch die Prüfziffer ersetzt. In der Abb. 19.17. ist das Unterprogramm aufgelistet. Man beachte, daß im Verbindungskapitel nur das vierstellige Feld definiert ist, dem das Argument des Hauptprogrammes entspricht. Im Unterprogramm ist dieses Feld mit CODE bezeichnet, das Argument im Hauptprogramm trägt aber den Namen CHECK. Mit Ausnahme des Eingangsnamens sind die Namen eines Unterprogrammes lokale Namen, d.h. sie sind nur im Unterprogramm bekannt.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  CHKDGT.
*
*
ENVIRONMENT DIVISION.
*
*
DATA DIVISION.
WORKING-STORAGE SECTION.
Ø1          W.
    Ø2      W1  PICTURE 9.
    Ø2      W2  PICTURE 9.
    Ø2      W3  PICTURE 9.
    Ø2      W4  PICTURE 9.
Ø1          X   PICTURE 99.
Ø1          Y     REDEFINES X.
    Ø2      Y1  PICTURE 9.
    Ø2      Y2  PICTURE 9.
LINKAGE SECTION.
Ø1          CODE.
    Ø2      C1  PICTURE 999.
    Ø2      C4  PICTURE 9.
*
*
PROCEDURE DIVISION  USING CODE.
    MOVE CODE TO W.
    SUBTRACT  W2 FROM 5Ø,  GIVING X.
    TRANSFORM  W FROM 'Ø123456789' TO 'Ø2468135Ø9'.
    SUBTRACT  W1, W3 FROM X.
    MOVE Y2 TO C4.
    GOBACK.
```

Anmerkung:
Die Spalten 1 bis 6 sind der Übersicht wegen nicht angegeben.

Abb. 19.17. Beispiel für ein Unterprogramm

In Abb. 19.18. ist ein Alternativprogramm zu dem in Abb. 19.17. gezeigten dargestellt. Bei ihm wird die ENTRY-Vereinbarung benutzt, um die Eingangsstelle ins Unterprogramm zu fixieren. Die Statementfolge im Hauptprogramm, die zum Aufruf des Unterprogrammes führt, bleibt jedoch bei dieser Codierung die gleiche.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  CHECKT.
*
*
 ENVIRONMENT DIVISION.
*
*
 DATA DIVISION.
*
 WORKING-STORAGE SECTION.
 Ø1          W.
     Ø2      W1  PICTURE 9.
     Ø2      W2  PICTURE 9.
     Ø2      W3  PICTURE 9.
     Ø2      W4  PICTURE 9.
 Ø1          X   PICTURE 99.
 Ø1          Y     REDEFINES X.
     Ø2      Y1  PICTURE 9.
     Ø2      Y2  PICTURE 9.
*
 LINKAGE SECTION.
 Ø1          CODE.
     Ø2      C1  PICTURE 999.
     Ø2      C4  PICTURE 9.
*
*
 PROCEDURE DIVISION.
     ENTRY 'CHKDGT'  USING CODE.
     MOVE CODE TO W.
     SUBTRACT  W2 FROM 5Ø  GIVING X.
     TRANSFORM  W FROM 'Ø123456789' TO 'Ø246813579'.
     SUBTRACT  W1, W3 FROM X.
     MOVE Y2 TO C4.
     GOBACK.
```

Anmerkungen:

1) Die primäre Eingangsstelle heißt CHECKT. Die sekundäre Eingangsstelle heißt CHKDGT, d.h. genau so wie die primäre Eingangsstelle des Unterprogrammes der Abb. 19.17.

2) Der Übersichtlichkeit wegen ist der Inhalt der Spalten 1 bis 6 nicht aufgeführt.

Abb. 19.18. Beispiel für ein Unterprogramm mit sekundärer Eingangsstelle

In Abb. 19.19. sind die Lochkartenpakete angedeutet, die notwendig sind, um die beiden Programmeinheiten (Hauptprogramm von Abb. 18.19., Unterprogramm von den Abb. 19.17. oder 19.18.) zu übersetzen, zu binden und schließlich das Gesamtprogramm auszuführen.

Betriebssystem DOS	Betriebssystem OS
// JOB CBL4	//CBL4 JOB ···
(Zuweisungs- und Kennsatzstatements, falls erforderlich)	// EXEC COBUFC,PARM.COB='LOAD,NODECK'
// OPTION LINK	//COB.SYSLIN DD DSN=&LOADSET,DISP=(MOD,PASS),
// EXEC FCOBOL	// UNIT=SYSDA,SPACE=(8Ø,(5ØØ,1ØØ))
(Quellenprogramm: *Hauptprogramm*)	//COB.SYSIN DD *
/*	(Quellenprogramm: *Hauptprogramm*)
// EXEC FCOBOL	// EXEC COBUFCLG
(Quellenprogramm: *Unterprogramm*)	//COB.SYSIN DD *
/*	(Quellenprogramm: *Unterprogramm*)
// EXEC LNKEDT	//GO.SYSØØ8 DD ···
// EXEC	//GO.SYSØØ9 DD ···
(Datenkarten)	//GO.SYSØ11 DD ···
/*	//GO.SYSØ1Ø DD *
/&	(Datenkarten)
	/*

Abb. 19.19. Zusammenstellung des Lochkartenpaketes zum Übersetzen, Binden und Ausführen der Programmeinheiten von Abb. 18.19. sowie Abb. 19.17 oder 19.18

Im Betriebssystem OS ist die Aufnahme des Statements

```
COB.SYSLIN   DD   ···
```

deshalb erforderlich, weil die von der IBM herausgegebene und gewartete Standardprozedur für die alleinige Übersetzung ohne dieses Statement nicht auskommt. Im übrigen genügt es, dieses Statement einfach aus dem Übersetzungsstep der Prozedur COBUCLG zu kopieren.

19.15. Segmentierung von Programmen

Es ist möglich, ein Programm zu übersetzen, das so umfangreich ist, daß es den für die Ausführung bereitstehenden Hauptspeicherplatz übersteigt (gilt nicht für die Variante VS2 des OS). Die COBOL-Kompilierer für das System IBM /370 ermöglichen die Überlagerung von Programmteilen während der Ausführung, so daß der gesamte Hauptspeicherbedarf reduziert wird, was natürlich auf Kosten der Ausführungszeit geht.

Um ein Programm zu segmentieren [1], muß der gesamte Prozedurteil in Kapitel unterteilt werden. Das bedeutet, daß die erste Zeile im Prozedurteil, die auf END DECLARATIVES folgt, einen Kapitelnamen enthalten muß. Dem Wort SECTION kann nach einem Leerzeichen eine Vorrangnummer im Wertebereich zwischen 0 und 99 einschließlich angehängt werden. Wird eine solche nicht niedergeschrieben, so wird die Vorrangnummer 0 angenommen. Bei allen Maßnahmen müssen die Vorrangnummern fehlen.

Die Verwendung der in Abschnitt 18.11. erwähnten Klausel SEGMENT-LIMIT[2] im Paragraphen OBJECT-COMPUTER des Ausrüstungskapitels des Maschinenteils ist freigestellt. Wird sie weggelassen, so wird für anz der Wert 50 angenommen. Unter anz ist die Segmentgrenze zu verstehen.

Das permanente, d.h. immer im Hauptspeicher befindliche Segment setzt sich aus allen Kapiteln zusammen, deren Vorrangnummer von 0 bis zu der um 1 verringerten Segmentgrenze reicht. Hinzu kommen alle Konstanten und Literale aus den Kapiteln mit einer höheren Vorrangnummer.

Für die Vorrangnummern zwischen der Segmentgrenze bis einschließlich 49 werden alle Kapitel mit dergleichen Nummer zu einem festen Überlagerunssegment zusammengefaßt. Und schließlich gilt für die Vorrangnummern, die größer als 49 sind, daß alle Kapitel mit der gleichen Vorrangnummer zu einem unabhängigen Segment zusammengefaßt werden.

Während der Programmausführung befindet sich das permanente Segment ständig im Hauptspeicher. Fast immer liegt außerdem ein anderes Segment im Hauptspeicher. Der gesamte Hauptspeicherbedarf ist gleich der Summe aus der Länge des permanenten Segmentes und aus der Länge entweder des größten Überlagerungssegmentes oder des größten unabhängigen Segmentes.

Für das Codieren segmentierter Programme gelten folgende Restriktionen:

1) Ein ALTER-Statement darf sich nicht auf ein unbedingtes Verzweigungsstatement (GO TO) beziehen, das in einem Kapitel vorliegt, dessen Vorrangnummer zwischen 50 und 99 einschließlich liegt; es sei denn, es befindet sich selbst in einem Kapitel mit der gleichen Vorrangnummer.
2) Eine aufgerufene Programmeinheit kann nicht segmentiert werden.

Für den Subsetkompilierer des Betriebssystemes DOS gilt weiterhin, daß PERFORM-Statements in unabhängigen Segmenten ihre Wirkungssphäre nicht auf Kapitel erstrecken dürfen, die zu anderen unabhängigen Segmenten gehören. In keinem Fall dürfen PERFORM-Anweisungen eine Wirkungssphäre aufweisen, die Kapitel aus mehr als einem Segment umfaßt.

Unterprogramme müssen in das permanente Segment eingeschlossen werden. Die Kompilierer des Betriebssystems OS sorgen automatisch dafür, die Kompilierer des DOS nicht. Im Betriebssystem DOS bedeutet diese Erfordernis einen manuellen Eingriff nach der Übersetzung

---

1) Unter einem Segment versteht man einen Teil eines Programmes; in COBOL ein aus einem oder mehreren Kapiteln bestehenden Programmteil.
2) Durch den Subsetkompilierer des Betriebssystems DOS wird diese Klausel nicht unterstützt.

und vor dem Binden. Dieser Eingriff entfällt, wenn das Unterprogramm nur in den Kapiteln aufgerufen wird, die zum permanenten Segment gehören.

Im Betriebssystem OS ist der Binder (Linkage Editor) mit dem OVLY-Parameter (Überlagerungsparameter) zu versorgen, wenn ein segmentiertes Programm gebunden werden soll.

Im Betriebssystem DOS werden Phasennamen generiert, die nicht vom Programmierer vergeben werden brauchen. Das permanente Segment besitzt als Namen die ersten sechs Zeichen des Programmnamens, der ja unter PROGRAM-ID vergeben ist. Die anderen Segmente weisen Namen auf, die aus dem Namen des permanenten Segments mit zwei angehängten Ziffern bestehen. Diese wiederum bezeichnen die Segment- (oder Kapitel-) Nummer.

19.16. Indexnamen

Jede Erklärung im Datenteil, die mit einer OCCURS-Klausel behaftet ist, kann darüberhinaus auch die Klausel

```
[INDEXED BY {indexname} ...]
```

besitzen. Bis zu 12 Indexnamen können angegeben werden, obgleich einer gewöhnlich ausreicht. Ein Indexname ist nach den Regeln zu bilden, die für Datennamen gelten, obgleich er nicht als solcher gilt. Der Indexname ist bereits vollständig definiert, wenn er in der INDEX-Klausel erscheint, jede weitere Erklärung erübrigt sich, d.h. er darf nirgendwo sonst im Datenteil auftreten. Für jeden im Programm benutzten Indexnamen richtet der Kompilierer ein Wort zur Aufnahme von Dualzahlen ein. Anfangswerte, das ergibt sich aus den zuvor gemachten Äußerungen, können natürlich nicht zugewiesen werden.

Indexnamen können als Indizes in Vergleichsausdrücken, im PERFORM-Statement und in den beiden bisher noch nicht behandelten Statements SET und SEARCH benutzt werden.

Bei der Betrachtung von Indexnamen sollten wir uns zuerst um den aktuellen Wert kümmern, der unter dem Indexnamen gespeichert ist. Mit diesem ist eine sogenannte Gliednummer verbunden. Die Gliednummer ist gleich einer natürlichen Zahl, durch die die Stellung eines bestimmten Gliedes innerhalb eines Bereiches (einer Tabelle) gekennzeichnet ist. Auf die Gliednummer muß man sich bekanntlich beziehen, wenn man ein bestimmtes Glied des durch die OCCURS-Klausel definierten Bereiches ansprechen will. Der Wert, der dem Indexnamen zugeordnet ist, ist gleich $(e-1)\cdot u$, wobei e die Gliednummer bedeutet und u die Länge in Bytes, die zu dem mit der OCCURS-Klausel versehenen Datennamen gehört. Was damit gemeint ist, kann man sich schnell klar machen: Der Wert, der einem Indexnamen zugewiesen ist, ist gleich dem Abstand des betreffenden Gliedes vom Bereichsbeginn an. Sehen wir uns ein Beispiel an:

```
Ø2    ABC     OCCURS 3  INDEXED BY BC.
  Ø3  AD   PICTURE X(4).
  Ø3  AE   PICTURE 9(5).
```

Wollen wir uns auf das 2. Glied der Gruppe ABC beziehen, so können wir ABC (2) schreiben. Benutzen wir stattdessen den eingeführten Indexnamen BC, so würden wir ABC (BC) zu schreiben haben, wobei BC den Wert 9 enthält. Für die im Beispiel genannte Tabelle mit drei jeweils neun Bytes langen Gliedern lauten die drei möglichen Werte für den Indexnamen 0, 9 und 18, die damit den Gliednummern 1, 2 und 3 entsprechen.

Ein neuer Wert wird einem Indexnamen durch das SET-Statement zugewiesen. Dieses lautet allgemein:

$$\underline{\text{SET}} \left\{ \begin{matrix} \underline{\text{indexname}} \\ \underline{\text{bez-1}} \end{matrix} \right\} \cdots \underline{\text{TO}} \left\{ \begin{matrix} \underline{\text{indexname-2}} \\ \underline{\text{bez-2}} \\ \underline{\text{literal-2}} \end{matrix} \right\}$$

Wird literal-2 benutzt, so muß dieses eine natürliche Zahl darstellen. Sowohl hinter bez-1 als auch hinter bez-2 dürfen sich nur numerische Datenelemente verbergen, die mit einem aus ganzen Zahlen bestehenden Wertevorrat erklärt sind. Die mit indexname-2 verbundene Elementnummer oder der Wert von literal-2 oder der Wert von bez-2 wird bez-1 zugewiesen. Ist indexname-1 statt bez-1 niedergeschrieben, so werden die genannten Werte als Gliednummer mit indexname-1 verbunden; anschließend erfolgt dann aus dieser Gliednummer die Ermittlung und Zuweisung des zu indexname-1 gehörenden Wertes.

Die SET-Anweisung kann auch in der folgenden zweiten Form codiert werden:

$$\underline{\text{SET}} \ \{\underline{\text{indexname-3}}\} \cdots \left\{ \begin{matrix} \underline{\text{UP}} \\ \underline{\text{DOWN}} \end{matrix} \right\} \ \underline{\text{BY}} \ \left\{ \begin{matrix} \underline{\text{bez-2}} \\ \underline{\text{literal-2}} \end{matrix} \right\}$$

Sie hat zur Folge, daß die mit indexname-3 verbundene Gliednummer um den Wert in der BY-Klausel erhöht (UP) oder erniedrigt (DOWN) wird.

Indexnamen können auch in Indexausdrücken an Stelle von Datennamen verwendet werden:

$$\underline{\text{indexname}} \ \left[\left\{ \begin{matrix} + \\ - \end{matrix} \right\} \ \underline{\text{gz}} \ \right]$$

Hierbei stellt gz eine natürliche Zahl dar. Ist diese codiert, so sprechen wir von einer indirekten Indizierung, fehlt diese, von einer direkten Indizierung. Die unter gz spezifizierte Zahl wird zu der Gliednummer addiert bzw. subtrahiert, die mit indexname verknüpft ist.

Als Indizes können wir entweder Datennamen und Literale oder Indexnamen benutzen, jedoch niemals beide in der gleichen Referenz. Somit ist z.B.

```
X (D,I)
```

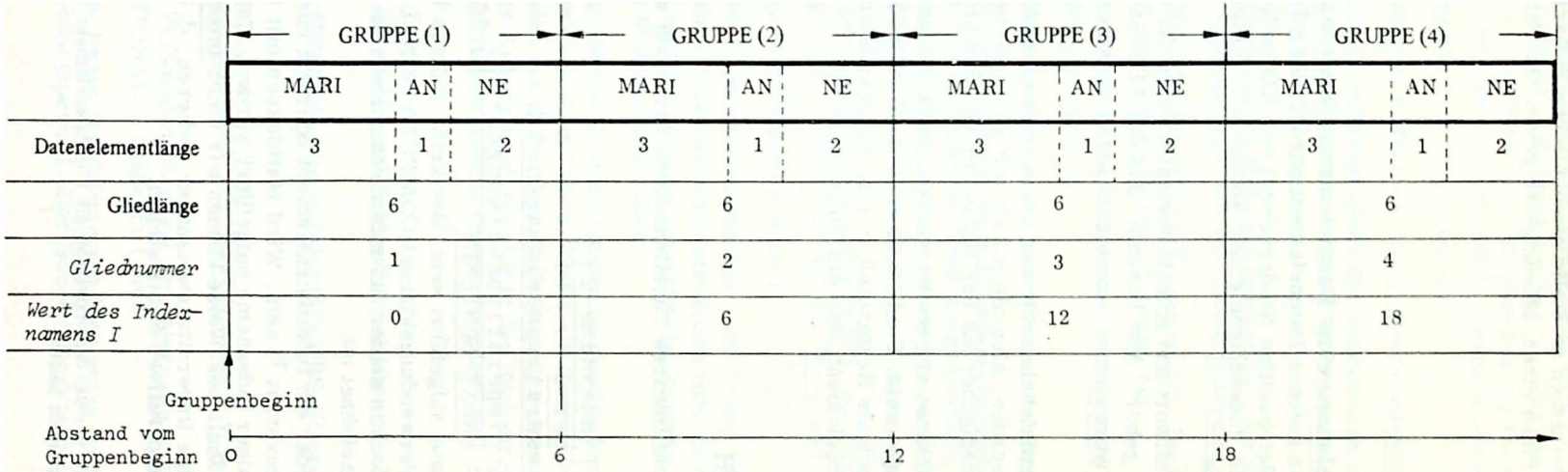
nicht akzeptabel, wenn D ein Datenname und I ein Indexname ist.

In Vergleichsbedingungen verbirgt sich hinter einem in den Vergleich eingehenden Indexnamen die mit ihm verbundene Gliednummer; der Vergleich erfolgt also auf numerischer Basis. Es ist deshalb nur gestattet, Indexnamen mit den folgenden anderen Größen zu vergleichen:

a) Indexnamen
b) Interne Dezimaldatenelemente
c) Externe Dezimaldatenelemente
d) Dualdatenelemente
e) Numerische Literale
f) Figurative Konstante ZERO

In PERFORM-Anweisungen kann ein Indexname an Stelle einer Bezeichnung in den Klauseln VARYING, FROM und AFTER benutzt werden. Dabei wird stets die mit dem Indexnamen verknüpfte Gliednummer benutzt.

Der Unterschied zwischen dem Wert eines Indexnamens und der mit diesem verknüpften Gliednummer wird in der Abb. 19.20. im Detail erläutert.



Der Skizze liegt folgende Erklärung zugrunde:

```
Ø2    GRUPPE   OCCURS 4 TIMES  INDEXED BY I.
  Ø3  MARI  PICTURE XXX.
  Ø3  AN    PICTURE X.
  Ø3  NE    PICTURE XX.
```

Abb. 19.20. Veranschaulichung der Werte von Indexnamen und der Gliednummern am Beispiel eines eindimensionalen Bereiches

Ein Vorteil bei der Benutzung von Indexnamen an Stelle gewöhnlicher Indizes entsteht mitunter daraus, daß der Kompilierer aus ihnen einen leistungsfähigeren Maschinencode erzeugen kann.

19.17. Indexdatenelemente

In der Verwendungsklausel (USAGE-Klausel) eines Datenelementes kann das Attribut INDEX niedergeschrieben werden. Ein solches Datenelement wird Indexdatenelement genannt. Nicht erlaubt ist dabei die gleichzeitige Niederschrift der Klauseln JUSTIFIED, PICTURE, BLANK und VALUE. Einem Indexdatenelement können alle Stufennummern mit Ausnahme von 88 zugeteilt werden.

Jedes Indexdatenelement belegt ein Vollwort und enthält normalerweise die Gliednummer, die zu irgendeinem Indexnamen gehört. Man beachte, daß das SET-Statement nicht benutzt werden kann, um einen Wert zu bzw. von einem Indexdatenelement zu addieren bzw. zu subtrahieren.

In Vergleichsbedingungen können Indexdatenelemente nur mit anderen Indexdatenelementen oder mit Indexnamen verglichen werden, niemals jedoch mit numerischen Datenelementen. Der Vergleich ist ein numerischer auf der Basis der Gliednummern.

Indexdatenelemente können nicht als Indizes verwendet werden, auch dürfen sie nicht Bestandteil von PERFORM-Anweisungen sein. Sie können auch nicht in MOVE-Statements aufgenommen werden. Gehen sie jedoch als Bestandteil in eine Gruppenübertragung ein, so verhalten sie sich wie ein Dualdatenelement, das ein Vollwort belegt.

19.18. Die Anweisung SEARCH

Durch die SEARCH-Anweisung kann ein lineares Tabellensuchen veranlaßt werden. Allgemein lautet sie wie folgt:

```
SEARCH  bez-1 [VARYING {indexname-2} ]
              {bez-2   }
              [AT END imperativanweisungen-1 ]
              {WHEN bed {imperativanweisungen-2} } ...
                        {NEXT SENTENCE          }
```

Hierbei muß unter bez-1 ein Datenname aufgeführt sein, der keine Indizes irgendwelcher Art aufweist und auch nicht mit der Verwendungsklausel COMPUTATIONAL-1 oder COMPUTATIONAL-2 erklärt ist, jedoch in seiner Definition sowohl mit der OCCURS- und mit der INDEXED-Klausel ausgezeichnet ist.

Wird bez-2 benutzt, so muß es entweder ein Indexdatenelement oder ein numerisches Datenelement mit ganzzahligem Wertevorrat [1] sein. Wird stattdessen mit indexname-2 operiert, so kann darunter ein beliebiger Indexname aufgeführt werden. Unter bed verstehen wir irgendeine Bedingung. In ähnlicher Weise können wir unter imperativanweisungen-1 bzw. imperativanweisungen-2 beliebige Imperativanweisungen codieren, d.h. es kann dort auch mehr als eine Anweisung imperativer Natur stehen.

---

1) Darunter versteht man ein externes dezimales, internes dezimales oder ein Dualdatenelement, das hinter dem Dezimalpunkt keine Ziffernstellen mehr besitzt.

Mit bez-1 sind ein oder mehrere Indexnamen verknüpft. Sei indexname-1 der erste Indexname in der INDEXED-Klausel von bez-1. Für die Ausführung der SEARCH-Anweisung wird nun ein Indexname benötigt. Nennen wir diesen einmal i-1. Ein zweiter Indexname kann in die Operation ebenfalls eingeschlossen sein, er möge i-2 heißen. Dann ergeben sich folgende vier Möglichkeiten:

1) Bei Weglassung von VARYING ist i-1 gleich indexname-1.

2) In der VARYING-Klausel ist einer der Indexnamen aufgeführt, die in der INDEXED-Klausel von bez-1 stehen. Dann ist i-1 gleich indexname-2, d.h. gleich dem Indexnamen in der VARYING-Klausel.

3) In der VARYING-Klausel steht ein Indexname, der nicht mit bez-1 verknüpft ist. Dann ist i-1 gleich indexname-1 und i-2 gleich indexname-2.

4) In der VARYING-Klausel ist bez-2 aufgeführt. Dann ist i-1 gleich indexname-1; außerdem geht auch bez-2 in die Operation ein.

Sei i-max der maximal mögliche Indexwert für bez-1. Dann läuft die Tabellensuchoperation in folgenden Schritten ab:

1. Wenn i-1 größer als i-max ist, so wird nach Schritt 5 verzweigt.
2. Auswertung der Bedingungen in der niedergeschriebenen Reihenfolge:
   Ist irgendeine der aufgeführten Bedingungen erfüllt, so wird der Suchvorgang beendet und die für diese Bedingung spezifizierten Imperativanweisungen ausgeführt. Danach geht die Programmablaufsteuerung zum nächsten Statementsatz über.
3. Ist keine der genannten Bedingungen erfüllt, so wird 1 zur Gliednummer von i-1 addiert. Die gleiche Addition erfolgt auf bez-2 und auf die Gliednummer von i-2.
4. Rückwärtsverzweigung zum Schritt 1.
5. Ausführung der Imperativanweisungen, die in der Klausel AT END enthalten sind. Danach geht die Programmablaufsteuerung zum nächsten Statementsatz über. Ist diese Klausel nicht vorhanden, so geht die Programmablaufsteuerung unmittelbar zum nächsten Statementsatz über.

Man merke sich, daß das lineare Suchen so durchgeführt wird, daß nur der im Statement angesprochene Index variiert wird. Weist ein Bereich mehr als eine Dimension auf, so muß für jeden Index ein getrennter Suchvorgang aufgesetzt werden. Es ist weiterhin zu beachten, daß der Anfangswert für den in den Suchvorgang einbezogenen Index vor Ausführung des SEARCH-Statements gesetzt werden muß.

19.19. Aufsteigende und fallende Schlüssel

Eintragungen im Datenteil, die mit der OCCURS-Klausel erklärt sind, können zusätzlich auch die Klausel

$$\left[\left\{\begin{matrix}\text{ASCENDING}\\\text{DESCENDING}\end{matrix}\right\}\ \text{KEY IS}\ \{\text{datenname}\}\ \cdots\right]$$

besitzen. In der Verwendungsklausel für datenname muß DISPLAY, COMPUTATIONAL oder COMPUTATIONAL-3 aufgeführt sein.

Ist datenname der Name des definierten Elementes oder der Gruppe selbst, dann ist er der einzige mögliche Operand. Der Bereich liegt in diesem Fall dann in richtiger Ordnung

vor, wenn er nach dem Inhalt von datenname in aufsteigender (ASCENDING) oder in fallender (DESCENDING) Folge sortiert ist. Doch sehen wir uns ein kurzes Beispiel an. Zunächst seien die Definitionen des Datenteils aufgeführt:

```
Ø2     B  OCCURS 3 TIMES DESCENDING B.
  Ø3   BA  PICTURE XX.
  Ø3   BB  PICTURE 999.
```

Die Elemente dieser Tabelle sind dann in richtiger Reihenfolge angeordnet, wenn sie nach dem Inhalt der Gruppe B selbst absteigend vorliegen. Dabei gelten natürlich die Regeln für den Vergleich von Gruppen. Somit wäre beispielsweise die Reihenfolge

```
WZ123
QY123
QY122
```

durchaus in Ordnung.

Ist datenname nicht gleich dem Namen des Elementes oder der Gruppe selbst, so gelten die folgenden Regeln:

1. Es dürfen nicht mehr als 12 Datennamen als Operanden in der Klausel aufgeführt sein.
2. Die Summe der Längen der Operanden darf 256 Bytes nicht überschreiten.
3. Die Operanden müssen in der Reihenfolge aufgeführt sein, die ihrer Wichtigkeit im gesamten Sortierbegriff zukommt, d.h. zuerst kommt der Hauptordnungsbegriff, dann der Reihe nach die Nebenordnungsbegriffe.
4. Jeder Operand muß dem definierten Bereich untergeordnet sein.
5. Kein Operand darf selbst durch die OCCURS-Klausel ausgezeichnet sein.
6. Kein Operand darf einer Gruppe innerhalb der definierten Gruppe untergeordnet sein, die mit der OCCURS-Klausel definiert ist.
7. Ist die Bereichslänge variabel, so muß jeder Operand vor dem variablen Teil des Bereiches auftreten.

Wir wollen diese Regeln an einem Beispiel demonstrieren. Die Datendefinition möge wie folgt lauten:

```
Ø2     B  OCCURS 3 TIMES DESCENDING BA BB.
  Ø3   BA  PICTURE XX.
  Ø3   BB  PICTURE 999.
```

Diese Beispiel führt zum gleichen Resultat wie das erste Beispiel. Wenn jedoch BB mit Vorzeichen erklärt wäre, könnten die Ergebnisse differieren, da der Vergleich der Werte der Elemente BB numerisch erfolgt.

Die Klauseln ASCENDING bzw. DESCENDING sind nur dann erforderlich und zugleich auch nützlich, wenn die mit ihnen definierte Größe (Gruppe oder Element) in das Statement SEARCH ALL eingeht.

19.20. Die Anweisung SEARCH ALL

Das Statement SEARCH ALL bewirkt ein nichtlineares Durchsuchen eines Bereiches. Seine allgemeine Form lautet:

```
SEARCH ALL bez-1
       [AT END imperativanweisungen-1]
           WHEN bed {imperativanweisungen-2 }
                    {NEXT SENTENCE          }
```

Durchsucht wird der Bereich mit der Bezeichnung bez-1 nach einem Element, das der Bedingung bed genügt.

Beim Statement SEARCH ALL muß also bez-1 der Name eines Bereiches sein, d.h. einer Eintragung mit den Klauseln OCCURS, INDEXED und ASCENDING. Selbstverständlich muß der Name selbst frei von Indizes sein. Außerdem darf er keine Gleitpunktdaten repräsentieren. Andererseits darf er selbst einer Gruppe untergeordnet sein, die die OCCURS-Klausel besitzt. Das Durchsuchen wird jedoch nur durch die Variierung des Index bewerkstelligt, der zu bez-1 gehört. Wie üblich verbergen sich hinter imperativanweisungen-1 und imperativanweisungen-2 ein oder mehrere Imperativanweisungen.

Die Bedingung bed unterliegt gewissen Einschränkungen, die ihrer Bedeutung gemäß nachstehend aufgeführt sind:

1. Wird unter bed ein Bedingungsname genannt, so muß die Eintragung mit der Stufennummer 88 einem Datennamen untergeordnet sein, der in der KEY-Klausel (ASCENDING bzw. DESCENDING) der Datenerklärung aufgeführt ist.
2. Ist bed eine Vergleichsbedingung, so dürfen nur die Vergleichsoperatoren = oder NOT EQUAL genannt werden, wobei einer der Operatoren ein Datenname sein muß, der in der KEY-Klausel steht.
3. Unter bed kann auch eine Verbundbedingung mit dem Operator AND genannt werden, vorausgesetzt, daß die in ihr enthaltenen einfachen Bedingungen die unter 1. und 2. aufgeführten Einschränkungen nicht verletzen.
4. Unter bed können beliebige Datennamen aus der KEY-Klausel geprüft werden, vorausgesetzt, daß alle ihm vorausgehenden Datennamen in der KEY-Klausel ebenfalls geprüft werden.

Dem Durchsuchen des Bereiches ist das Ziel gesteckt, ein Bereichselement zu finden, das der aufgeführten Bedingung bed genügt. Erfüllt kein Element diese Bedingung, so werden die in der Klausel AT END enthaltenen Imperativanweisungen ausgeführt, falls diese vorhanden sind. Fehlen sie, so geht selbstverständlich die Programmablaufsteuerung zum nächsten Statementsatz über. Ist die Bedingung bed erfüllt, so wird dem ersten zu bez-1 gehörenden Indexnamen der Wert zugewiesen, der auf das gefundene Bereichselement zeigt. Anschließend werden die unter imperativanweisungen-2 genannten Anweisungen ausgeführt, bevor die Programmablaufsteuerung zum nächsten Satz übergeht.

Bei einem erfolglosen Durchsuchen kann der Wert des ersten Indexnamens von bez-1 nicht vorhergesagt werden. Erfüllt mehr als ein Element des Bereiches die Bedingung bed, so gibt es keine Möglichkeit, welches Element den Suchvorgang beendete, da eben das Durchsuchen nicht in linearer Reihenfolge, sondern nach einem anderen, den Herstellern von COBOL-Kompilierern überlassenen Reihenfolge erfolgt. [1)]

---

1) Meist wird hierzu das „binäre Tabellensuchen", auch Tabellensuchen durch Springen genannt, eingesetzt. Die Zahl der Suchschritte ist hier gleich

$$5 = g(_2\log n) + 1$$

wobei n die Anzahl der Tabellenelemente ist und g (x) der ganzzahlige Anteil von x.

19.21. Die Handhabung von Ketten

Die in diesem Abschnitt zu besprechenden Anweisungen sind nicht verfügbar, wenn die Kompiliererversionen 2 und 3 für die Übersetzung herangezogen werden.

Zwei Anweisungen ermöglichen die Handhabung von Zeichenketten. Ihre allgemeine Form ist in der Abb. 19.21. aufgeführt.

```
I.    STRING  el-1 [el-2] ···  DELIMITED BY {el-3} ···
                                            {SIZE}
            INTO bez-1
           [WITH POINTER bez-2]
           [ON OVERFLOW imperativanweisungen]

II.   UNSTRING  bez-3
            [DELIMITED BY [ALL] el-4 [OR] [ALL] [el-5] ···]
             INTO  bez-4 [DELIMITER IN bez-5] [COUNT IN bez-6] ··
            [WITH POINTER bez-7]
            [TALLYING IN bez-8]
            [ON OVERFLOW imperativanweisungen]
```

Abb. 19.21. Allgemeine Form der Anweisungen zur Handhabung von Zeichenketten

Die in der Abb. 19.21. verwendeten Begriffe haben dabei folgende Bedeutungen:

Jedes der Elemente el-1, el-2, ··· stellt ein nichtnumerisches Literal, eine figurative Konstante (mit Ausnahme von ALL) oder eine Bezeichnung dar. Handelt es sich um eine Bezeichnung, so muß diese zu einem Datenelement mit der Verwendungsklausel DISPLAY gehören. Zusätzlich gilt für el-3, daß es keine Klausel OCCURS ··· DEPENDING ON ··· besitzen darf. Bei der Niederschrift von el-4 und el-5 ist zu beachten, daß es sich bei beiden um Gruppen oder Datenelemente handelt, die zur alphamerischen Kategorie gehören.

Unter bez-1, bez-2, ··· ist jeweils eine Bezeichnung zu verstehen. Sowohl bez-3 als auch bez-5 müssen Gruppen oder Datenelemente alphamerischer Natur sein. Die anderen Bezeichnungen dürfen nur Datenelemente sein. Zusätzlich ist zu beachten, daß in der Abbildung von bez-1 nur die Abbildungszeichen S, 9, V, A und X auftreten dürfen. Besitzt bez-1 die Klausel SIGN IS SEPARATE CHARACTER, so wird diese ignoriert. Als weitere Regel kommt hinzu, daß bez-2, bez-6, bez-7 und bez-8 nur Datenelemente repräsentieren dürfen, deren Wertevorrat ganzzahlig ist, d.h. bei ihnen darf in der Abbildung kein Abbildungszeichen 9 rechts vom Dezimalpunkt vorhanden sein. Die Abbildung von bez-4 wiederum darf kein Abbildungszeichen P enthalten. Im allgemeinen muß das mit bez-6 bezeichnete Datenelement fähig sein, die um 1 vergrößerte Länge von bez-4 aufzunehmen; für bez-7 gilt das gleiche bezüglich der Länge von bez-3. Und schließlich muß in bez-8 die Anzahl der Datenelemente mit der Bezeichnung bez-4 hineinpassen, die im Statement UNSTRING aufgeführt sind.

Bis zu 14 Operanden el-5 können aufgeführt werden. Keine Bezeichnung darf die Stufennummer 88 besitzen. Die Klauseln DELIMITER IN und COUNT IN sind nur dann erlaubt, wenn DELIMITED BY codiert ist. Alle eingesetzten figurativen Konstanten werden als nichtnumerische Literale betrachtet, die aus einem Zeichen bestehen.

Doch wenden wir uns von den Begriffen und den für sie geltenden Regeln ab und den Statements selbst zu. Das Statement STRING bewirkt die Verkettung von el-1, el-2, ··· zu einer Kette, die bez-1 als Wert zugewiesen wird. Wird dabei die Klausel DELIMINTED BY SIZE benutzt, so ist die Anzahl der Bytes, die von el-1 in die Verkettung einbezogen werden, gleich der Länge von el-1. Dasselbe trifft auch auf el-2, ··· zu. Das bedeutet also, daß der Inhalt des ersten Bytes von el-2 in das Byte von bez-1 gebracht wird, das als nächstes auf das letzte von el-1 belegte Byte folgt. Die Übertragung nach bez-1 beginnt beim ersten Byte von bez-1, es sei denn, daß die POINTER-Klausel niedergeschrieben wird. In diesem Fall fängt die Übertragung bei der Stelle von bez-1 an, die durch bez-2 festgelegt ist. Nach jeder Übertragung eines Zeichens in bez-1 wird der Wert von bez-2 um 1 erhöht. Wird die Klausel

DELIMITED BY el-3

niedergeschrieben, so kommt die Verkettung vorzeitig zum Ende, wenn eine gewisse Vergleichsbedingung erfüllt ist. Vor dem Übertragen des nächsten Bytes aus el-1, el-2, ··· nach bez-1 wird nämlich jeweils geprüft, ob die nächsten (zu übertragenden) Bytes den gleichen Inhalt wie el-3 besitzen, wobei die Länge von el-3 maßgebend für den Vergleich ist. Nach einer solchen frühzeitigen Beendigung der Übertragung der Zeichen eines Elementes wird die Verkettung mit dem nächsten Element el-2 fortgesetzt. Kann bez-1 kein weiteres Zeichen mehr aufnehmen, d.h. der Zähler hat die um 1 vergrößerte Länge von bez-1 erreicht, sind jedoch noch weitere Zeichen zu übertragen, so unterbleibt die Übertragung und es werden die in der OVERFLOW-Klausel stehenden Imperativanweisungen ausgeführt. Die Statements in dieser Klausel kommen auch dann zum Zuge, wenn bez-2 zu Beginn der Verkettungsoperation einen Wert kleiner als 1 oder größer als die Länge von bez-1 enthält. Die nicht von der Verkettung betroffenen Bytes von bez-1 bleiben unverändert erhalten. Der Inhalt aller in die Übertragung einbezogenen Bytes wird als alphamerisches Zeichen betrachtet.

Gehen wir nun zur Beschreibung der Entkettung über. Ist die Klausel DELIMITED BY weggelassen, so wird der Inhalt von bez-3 in das erste der Elemente bez-4 übertragen und zwar solange, bis das erste bez-4 gefüllt ist. Die Übertragung wird danach fortgesetzt, indem die Inhalte der nächsten noch nicht von der Übertragung berührten Bytes von bez-3 in das darauffolgende Datenelement bez-4 hinübergeschafft werden. Dieser Prozeß wird solange weiterverfolgt, bis entweder alle Bytes von bez-3 in die Übertragung einbezogen waren oder alle Elemente bez-4 gefüllt sind. Bei Spezifizierung von bez-7 beginnt die Auflösung von bez-3 bei der Stelle, auf die der Wert von bez-7 verweist, in allen anderen Fällen selbstverständlich beim ersten Byte von bez-3. Nach jeder Übertragung eines Bytes von bez-3 wird der Wert von bez-7 um 1 erhöht. Ist am Ende der Operation der Wert von bez-7 nicht gleich der um 1 erhöhten Länge von bez-3, so werden die Imperativanweisungen ausgeführt, die in der Klausel ON OVERFLOW stehen. Diese Klausel kommt auch dann zum Zuge, wenn der Anfangswert von bez-7 kleiner als 1 oder größer als die Länge von bez-3 war. Wird die Klausel TALLYING niedergeschrieben, so wird zum Wert von bez-8 immer dann eine 1 addiert, wenn ein neues Element mit der Bezeichnung bez-4 bei der Entkettung von bez-3 zum Zuge kommt. Ist der Anfangswert von bez-8 gleich Null, so ist am Schluß der Entkettungsoperation der Wert von bez-8 auf die Zahl der Elemente bez-4 angewachsen, in die Zeichen von bez-3 übertragen worden sind.

Bei der Übertragung der Zeichen von bez-3 nach den Datenelementen mit der Bezeichnung bez-4 ist die Anzahl der in die Bewegung einbezogenen Bytes gleich der Summe der Abbildungszeichen A, X und 9 in den Abbildungen der Elemente bez-4. Wenn notwendig werden numerische Felder mit Nullen, nichtnumerische mit Leerzeichen aufgefüllt.

Wird die Klausel DELIMITED BY niedergeschrieben, so erfolgen vor der Übertragung des nächsten Zeichens aus bez-3 Vergleiche mit den Zeichen, die in den Elementen mit den Bezeichnungen el-4, el-5, ··· stehen. Die Vergleiche erfolgen in der Reihenfolge, in der die Vergleichselemente im Statement aufgeführt sind und mit den Längen von el-4, el-5, ··· . Führt irgendeiner dieser Vergleiche auf das Resultat gleich, so wird die Übertragung nach bez-4 an dieser Stelle beendet, der Transport der gleichen Zeichen unterbleibt. Die Gesamtzahl der Zeichen, die nach dem gerade als Zielfeld fungierenden Datenelement bez-4 übertragen wurden, wird nach bez-6 hineingestellt. Die Zeichen von bez-3, die gleich den Zeichen von el-4, el-5, ··· waren, werden nach bez-5 übertragen, bzw. so viele, wie in bez-5 hineinpassen. Und schließlich wird bez-7 auf die Stelle von bez-3 gesetzt, die das erste Zeichen enthält, das auf die gleichen Zeichen folgt; nebenbei gesagt, werden diese auch Begrenzungszeichen genannt. Die eigentliche Entkettungsoperation wird nach der durch diese Teiloperationen bedingten Unterbrechung fortgesetzt, wobei als neues Zielfeld der Entkettung das nächste Datenelement bez-4 herangezogen wird. Wird vor einem der Elemente el-4, el-5, ··· das Wort ALL codiert, so wird das mögliche mehrfache Auftreten der Werte von el-4, el-5 im Rest von bez-3 als einfaches Auftreten angesehen. Alle Wiederholungen der Werte von el-4, el-5, ··· werden jedoch nach bez-5 gestellt. Es ist denkbar, daß mehrere gleiche Gruppen von Begrenzungszeichen unmittelbar aufeinander folgen. In diesem Fall können ein oder mehrere Felder mit den Bezeichnungen bez-4 zu inhaltslosen Feldern werden. Sie enthalten dann in Abhängigkeit von ihrem Datentyp Nullen oder Leerzeichen.

Es sei abschließend betont, daß die Verwendung der Statements Übertragungen zwischen alphamerischen Datenelementen beschränkt ist.

Aufgabengruppe 53

1) Ändere das Programm von Aufgabe 5), Aufgabengruppe 50 so ab, daß es als Unterprogramm eingesetzt werden kann.

2) Baue das Unterprogramm von Aufgabe 1) in ein Programm ein, das dazu bestimmt ist, Lochkarten zu lesen, die das Datum in der Form ttmmjj auf den Spalten 1 bis 6 enthalten und alphamerische Informationen auf den Spalten 7 bis 80. Das Programm soll neue Karten stanzen, die auf den Spalten 7 bis 80 die gleichen alphamerischen Informationen wie die Eingabekarten enthalten, auf den Spalten 1 bis 6 jedoch ein Datum, das um eine Woche später liegt als das eingelesene Datum. Wird also z.B. 130568 eingegeben, so ist 200568 zu stanzen, bei der Eingabe von 260268 ist 040368 zu stanzen (Schaltjahr!).

3) Die Aufgabe 1) der Aufgabengruppe 51 ist wie folgt zu erweitern:

   Es ist ein Unterprogramm zu schreiben, mit dem der eingegebene und E zugewiesene Text auf die Einhaltung gewisser grammatikalischer Regeln überprüft werden kann. Es ist sicherzustellen, daß genau ein Leerzeichen zwischen Wörtern und zwei am Ende eines jeden Satzes vorhanden sind. Der richtig gestellte und mit Groß- und Kleinbuchstaben versehene Text ist am Schluß des Unterprogrammes wiederum E zuzuweisen.

4) Unter Benutzung des Ergebnisses von Aufgabe 3) ist ein Programm zu schreiben, das eine Aufstellung druckt, aus der die Häufigkeit des Auftretens jedes einzelnen der 256 möglichen Zeichen des EBCDI-Codes im Text zu ersehen ist. Das Drucken von Zeilen mit Zeichen, die im Text nicht auftreten, soll unterbleiben.

Jede einzelne Druckzeile soll der Reihe nach enthalten:

— Sedezimalverschlüsselung des Zeichens
— Zeichen
— Häufigkeit

5) Zur Verwendung in Aufgabe 4) ist folgendes Unterprogramm zu schreiben:

Es ist der Wert eines einstelligen Datenelementes in dessen zweistelligen Sedezimalwert umzuwandeln.

6) Regional organisierte Dateien (auch als gestreut oder direkt organisierte Dateien bezeichnet) werden sehr oft dann eingerichtet, wenn Materialbestände zu führen sind und die Teilenummern zu lang sind. Nehmen wir zur Erläuterung einmal an, daß eine Firma mit achtstelligen Teilenummern arbeitet und nur 1500 Teile insgesamt zu verwalten hat. Wenn wir eine neue Teilenummer erhalten, so dividieren wir sie durch, sagen wir, 1699 (irgendeine Primzahl etwas größer als 1500) und verwenden den Rest als Schlüssel. Ist dieser Schlüssel bereits in Benutzung, so addieren wir eine 1 auf den Rest und versuchen es erneut. Irgendwann finden wir durch dieses Verfahren einmal einen Schlüssel, der noch nicht in Benutzung ist (notfalls müssen wir statt zum Addieren zum Subtrahieren übergehen). Den so gefundenen Schlüssel ordnen wir dem Materialteil zu und schreiben den entsprechenden Teilesatz.

Schreibe ein Programm, das Lochkarten einliest, auf denen die Teilenummern in den Spalten 1 bis 8 abgelocht sind und suche den entsprechenden Teilesatz auf, falls ein solcher vorhanden ist. Wenn für die angegebene Teilenummer kein Satz existiert, so ist eine Fehlermeldung zu drucken; im anderen Fall ist der zugehörige Teilesatz zu drucken.

Nach Voraussetzung sind die Teilesätze 80 Stellen lang.

20. Die Assemblierersprache

20.1. Das Codieren mit veränderlichen Adressen

Augenscheinlich erfordert die Codierung eines umfangreichen Problems in der Maschinensprache einen erheblichen Aufwand an Überlegung. Viele Menschen sind mit derartigen in die Einzelheiten gehenden Arbeiten nicht vertraut, so daß es nicht verwunderlich ist, wenn beim Programmieren Irrtümer unterlaufen. Wenn ein Programm mehr als ca. 50 Instruktionen umfaßt, ist es auch höchst ungewöhnlich, daß es bereits beim ersten Male korrekt abläuft. Bis zum Jahre 1951 etwa waren die meisten Bemühungen beim Programmieren dieser geistigen Arbeit gewidmet. Bedenkt man nun, daß viele Programme aus weit mehr als tausend Instruktionen bestehen, so braucht man nicht darüber erstaunt zu sein, wenn sich die besten Programmierer schon ziemlich frühzeitig um die Entwicklung von Methoden kümmerten, die ihre Anstrengungen vereinfachen konnten.

Bisher haben wir unsere Programme in einer Form geschrieben, in der die Instruktionen direkt von der Maschine verstanden und interpretiert werden konnten. Die Sprache, die dieser Form der Codierung zugrundeliegt, bezeichnet man als Maschinensprache oder absolute Sprache, manchmal wird sie auch Computergrundsprache genannt. Bei dieser Art des Programmierens ist es wohl einleuchtend, wenn man damals als wertvollstes Arbeitswerkzeug des Programmierers zum Scherz den Radiergummi erwähnte. Um 1952 erkannte man dann, daß man das Radieren und Ändern erheblich einschränken konnte, wenn man auf das Niederschreiben absoluter Adressen verzichtete, bevor man nicht das ganze Programm zu Papier gebracht hatte. Zum Vergleich sollte man das nachstehende Programm betrachten; einmal ist es in der Maschinensprache geschrieben, das zweite Mal mit veränderlichen Adressen (Abb. 20.1.).

Maschinensprache			Veränderliche Adressen		
ØØ2 ØØØ	BALR	Ø5 B Ø		Ø5	r1 Ø
ØØ2 ØØ2	L	58 5 Ø B Ø98		58	r2 a
ØØ2 ØØ6	S	5B 5 Ø B 1Ø2	b	5B	r2 c
ØØ2 ØØA	BC	47 2 Ø B ØØ4		47	2 b
ØØ2 ØØE	ST	5Ø 5 Ø B Ø98		5Ø	r2 a

Abb. 20.1. Sprachvergleich

Man vermied nicht nur das Niederschreiben absoluter Adressen in den Instruktionen, sondern man verzichtete auch darauf, die Instruktionsadressen selbst anzugeben, es sei denn, man muß sich an einer anderen Stelle des Programmes auf sie beziehen. Somit wurde bei unserem obigen Beispiel nur der dritten Instruktion ein Name gegeben, da diese die einzige ist, die von den anderen Instruktionen angesprochen wird.

Nach der so erfolgreichen Niederschrift und der Ausmerzung aller gefundenen Fehler ging der Programmierer zum Programmanfang zurück und trug die geeigneten absoluten Adressen der Speicherplätze nach. Man fand überdies schließlich heraus, daß das Locherbedienungspersonal oft in der Lage war, den Austausch mnemonischer Codes gegen absolute Zahlen selbst vorzunehmen. Man ging deshalb danach noch einen Schritt weiter, indem man nicht nur veränderliche Adressen niederschrieb, sondern darüberhinaus sie auch noch mit den mnemonischen Operationsschlüsseln kombinierte (Abb. 20.2.).

```
Symbolische Codierung
        ¦ BALR ¦ r1   Ø
        ¦ L    ¦ r2   a
        ¦ S    ¦ r2   c
        ¦ BC   ¦ 2    b
        ¦ ST   ¦ r2   a
```

Abb. 20.2. Codierung mit symbolischen Operationsschlüsseln und Adressen

Es war nicht übermäßig schwierig, ein Programm in dieser symbolischen Form zu schreiben und es anschließend noch einmal vorzunehmen, um die aktuellen Adressen und Operationsschlüssel einzutragen. Im besonderen wurde es allmählich üblich, für die symbolischen Adressen einen Namen zu wählen, der mit der Größenbezeichnung selbst übereinstimmte oder zumindest ihr ähnelte, z.B. NETTO, STEUER, SUM usw. Vor dem Zurückgehen und dem Eintragen der absoluten Speicherplatzadressen legte sich der Programmierer gewöhnlich eine Symboltabelle an, in der jeder symbolische Namen und die ihm entsprechende aktuelle Adresse aufgeführt waren. Stieß er dann beim Durchsuchen seines Programmes auf einen besonderen Namen, brauchte er diesen nur durch die absolute Adresse zu ersetzen. Natürlich wurden die Register und die Hauptspeichergebiete für die Daten getrennt den Symbolen zugeordnet. Die Adressen wurden gewöhnlich unter Benutzung einer Basisadresse ausgedrückt, so daß das Programm mit nur wenigen Änderungen in einen beliebigen Teil des Hauptspeichers geschoben werden konnte. Der Programmierer konnte ebenso auch die echten Operationsschlüssel an Stelle der mnemonischen Schlüssel anfügen, doch überließ er diese Arbeit gewöhnlich dem Bedienungspersonal der Locher.

Die Ersetzung der mnemonischen Operationsschlüssel durch die echten numerischen Schlüssel erfordert nur sehr geringe geistige Anstrengungen. Man braucht sich dazu nämlich nur eine Tabelle anzulegen, in der die symbolischen Schlüssel und die ihnen entsprechenden numerischen Schlüssel aufgeführt sind, und diese zur Hand zu nehmen. Nun besteht aber überhaupt kein Grund dafür, daß diese Arbeit nicht auch von einem Computer übernommen werden kann, wenn man nur ein entsprechendes Programm für diese Aufgabe schreibt. Wenn weiterhin ein Programm nicht zu viele verschiedene symbolische Adressen verwendet, so ist es nicht zu schwierig, das Programm so zu schreiben, das es eine symbolische Instruktion nach der anderen einliest und diesen Instruktionen der Reihe nach aufeinanderfolgende Speicherplätze zuordnet, wobei selbstverständlich die Anzahl von Bytes freigehalten wird, die die betreffende Instruktion belegt. Das erfordert freilich den Aufbau einer Tabelle von Gleichheitsbeziehungen, z.B. a=002100, b=002006 usw., wobei jeder Platz auch relativ zu einem Basisregister ausgedrückt wird. Immer, wenn ein Symbol auftritt, sieht der Computer in dieser Tabelle nach und ordnet dann die geeignete Adresse zu.

Diese sogenannte automatische Programmierung kam im Jahre 1953 auf. Das in symbolischer Form geschriebene Programm, auch Quellenprogramm genannt, wurde Instruktion auf Instruktion in den Computer eingelesen, und das im Computer gespeicherte Programm, kurz Assemblierer genannt, stanzte für jede eingelesene Instruktion ihr Äquivalent in der Maschinensprache aus [1]. Das auf diese Weise entstandene Maschinenprogramm heißt Objektprogramm; dieses kann dann jederzeit, wenn seine Ausführung gewünscht wird, wieder in die Maschine geladen werden.

1) Die Assembliersprache gehört zur Klasse der maschinenorientierten Sprachen. Im angloamerikanischen Sprachraum wird die Assembliersprache als Assembler Language, abgekürzt AL, bezeichnet. Davon herrührend spricht man im Deutschen oft ebenfalls von der Assemblersprache.

Zwischen einem Assemblierer und einem Komplierer besteht folgender Unterschied: Für jede symbolische Instruktion des Quellenprogrammes gibt ein Assemblierer eine einzige Instruktion ins Objektprogramm aus, während ein Kompilierer (gleichgültig, ob für die FORTRAN-Sprache, für COBOL oder für PL/1) aus einer Anweisung des Quellenprogrammes eine ganze Reihe von Instruktionen generiert. Das bedeutet natürlich, daß wir im allgemeinen erwarten können, daß wir mit einer Anweisung (Kompilierer) mehr Operationen hervorrufen können als mit einer Instruktion (Assemblierer). Wir bezahlen andererseits bei Benutzung eines Kompilierers aber auch einen gewissen Preis dafür, daß wir nicht soviel zu codieren brauchen: Wir haben nämlich keine Übersicht über die ins Objektprogramm aufgenommenen einzelnen Instruktionen. Schreiben wir dagegen ein symbolisches Programm, so können wir jede Instruktion individuell verfolgen. Das bedeutet aber nichts anderes, als daß beim symbolischen Programmieren Dinge realisiert werden können, die durch Kompilierer nicht bewerkstelligt werden können, z.B. bestimmt maschinenspezifische Operationen; außerdem können wir bei großer Geschicklichkeit wirkungsvollere Programme schreiben, d.h. Programme, die statisch und dynamisch kürzer sind als die, die von Kompilierern generiert werden. Bei der IBM /370 sollte die Programmierung in der AL im allgemeinen nur dann erfolgen, wenn eine von der Regel abweichende Eingabe bzw. Ausgabe vorgesehen ist (Beispiele hierfür sind das Lesen von Dualkarten, das Lesen optisch abgetasteter Belege usw.), oder wenn der verfügbare Kompilierer ein Programm erzeugt, das zuviel Speicherplatz erfordert (eine zu große statische Länge besitzt) oder eine zu große Ausführungszeit benötigt (also eine zu große dynamische Länge besitzt). Die AL wird aber auch vielfach dazu benutzt, Unterprogramme zu schreiben, die in Verbindung mit Programmen verwendet werden, die von Kompilierern generiert werden.

20.2. Ein Beispiel

Wir wollen bestimmen, auf welchen Endwert ein Kapital von 1.-- DM anwächst, wenn es mit einem Zinssatz von p % verzinst wird. Wenn wir $i=\frac{p}{100}$ einführen, dann können wir den Endwert nach der Formel

$$A = 1 \cdot (1+i)^n$$

ermitteln. Die Anzahl der Zinsperioden ist durch n gegeben. Abb. 20.3. zeigt den Programmablaufplan für ein Programm, das die Werte von n und i einliest und das eine Tabelle ausdruckt, die sowohl die Anzahl jeder Zinsperiode als auch den Endwert nach jeder Zinsperiode enthält; j soll der Reihe nach die Werte $1, 2, \cdots, n-1, n$ annehmen. Die Ausgabe soll in folgender Form geschehen:

```
bbb1bbbb1.Ø35ØØØ 1)
   2     1.Ø71225
```

Bevor wir die Instruktionen niederschreiben, wollen wir die benötigten Hauptspeichergebiete zuordnen. Sowohl für den Betrag als auch für die kleinste Anzahl der Zinsperioden müssen wir Hauptspeicherplätze mit einem konstanten Inhalt bereitstellen; schließlich ist auch noch eine Schablone zur Druckaufbereitung erforderlich. Wir wollen die Konstanten gleichzeitig mit symbolischen Namen benennen:

```
AUF     4Ø2Ø2Ø2Ø 224Ø2Ø2Ø 214B2Ø2Ø 2Ø2Ø2Ø2Ø
EINS    ØØ1+
KAP     ØØ1ØØØØØØ+
```

Bei Programmen in der AL ist es relativ einfach, die Eingabe/Ausgabe-Operationen im Programm anzusprechen. Da wir jedoch diese Methode erst später behandeln werden, wollen wir uns momentan noch mit der Verwendung des im Abschnitt 4.5. besprochenen Verfahrens zur Praktizierung der EA begnügen. Wenn wir in die Kartenspalten 1 bis 3 den Wert von n und in die Kartenspalten 4 bis 9 den Wert von i ablochen, so benötigen wir vom 80stelligen Karteneingabegebiet nur die ersten 9 Bytes; die restlichen 71 Bytes bleiben unbenutzt. Der relevante Teil des Eingabegebietes sieht daher wie folgt aus:

```
N       XX XX XX.
I      .XX XX XX XX XX XX
```

Durch den Punkt wird hier die Stellung des Dezimalpunktes angezeigt.

Das Ausgabegebiet erfordert eigentlich nur 16 Bytes, wir wollen es mit DRU benennen. Da wir einen Zeilendrucker benutzen wollen, lassen wir anschließend 104 Bytes frei, so daß wir damit eine Druckzeile von 120 Stellen abdecken. Von diesen sind damit die letzten 104 Bytes unbenutzt. Ihnen sind deshalb als Anfangswerte Leerzeichen zuzuweisen.

Wir benötigen außerdem einige Arbeitsgebiete (Abb. 20.4.).

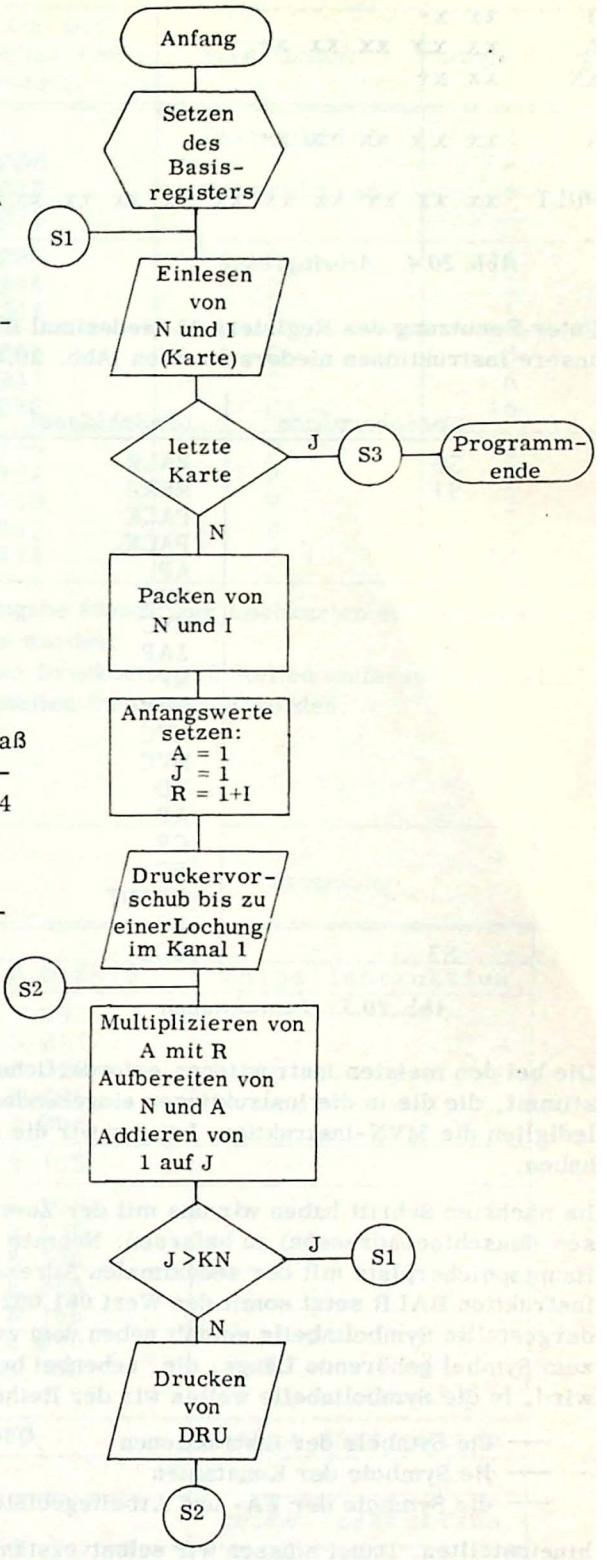


Abb. 20.3. Programmablaufplan

```
J       xx x+                                   (Gepacktes Feld)
A       xx x.x xx xx x+                         (Gepacktes Feld)
KN      xx x+                                   (Gepacktes Feld zur Aufnahme des
                                                eingelesenen Wertes von N)
R       xx x x xx xx x+                         (Gepacktes Feld zur Aufnahme von
                                                R=1+1)
MULT    xx xx xx xx xx xx xx xx xx xx 10 Bytes großes Feld zur Aufnahme
                                                des Produktes von A und R)
```

Abb. 20.4. Arbeitsgebiete

Unter Benutzung des Registers 11 (sedezimal B) als Basisregister können wir nunmehr unsere Instruktionen niederschreiben (Abb. 20.5.).

Speicherplatz	*OP-Schlüssel*	*Operanden*
SØ	BALR	B,Ø
S1	READ	N,S3
	PACK	KN,N
	PACK	R,I
	AP	R,KAP
	MVC	A,KAP
	MVC	J,EINS
S2	ZAP	MULT,A
	MP	MULT,R
	MVN	MULT+6(1),MULT+9
	MVC	A,MULT+2
	MVC	DRU,AUF
	ED	DRU,J
	AP	J,EINS
	CP	J,KN
	BC	2,S1
	PRINT	DRU
	BC	15,S2
S3	EOJ	

Abb. 20.5. Instruktionen

Die bei den meisten Instruktionen erforderliche Längenangabe wird aus den Längen bestimmt, die die in die Instruktionen eingehenden Felder besitzen. Eine Ausnahme bildet lediglich die MVN-Instruktion, bei der wir die Länge explizit in Klammern angegeben haben.

Im nächsten Schritt haben wir uns mit der Zuordnung der aktuellen Hauptspeicheradressen (Maschinenadressen) zu befassen. Nehmen wir hierzu an, daß das Programm auf dem Hauptspeicherplatz mit der sedezimalen Adresse 010 000 (dezimal 65536) beginnt. Die Instruktion BALR setzt somit den Wert 001 002 ins Register 11. Unsere in Abb. 20.6. dargestellte Symboltabelle enthält neben dem zugeordneten Hauptspeicherplatz auch die zum Symbol gehörende Länge, die, nebenbei bemerkt, oft als Längenattribut bezeichnet wird. In die Symboltabelle wollen wir der Reihe nach

- die Symbole der Instruktionen
- die Symbole der Konstanten
- die Symbole der EA- und Arbeitsgebiete

hineinstellten. Dabei müssen wir selbstverständlich nicht die Gebiete lückenlos aneinander angrenzen lassen.

Symbol	*Sedezimale Hauptspeicheradresse*	*Basisregister und Distanzadresse (relative Adresse)*		*Bemerkungen*	*Länge in Bytes*
SØ	Ø1Ø ØØØ	-	-	Instruktion	2
S1	Ø1Ø ØØ2	B	ØØØ	-	-
S2	Ø1Ø Ø3Ø	B	Ø2E	Instruktion	6
S3	Ø1Ø Ø78	B	Ø78	-	-
AUF	Ø1Ø Ø88	B	Ø86	C	16
EINS	Ø1Ø Ø98	B	Ø96	P	2
KAP	Ø1Ø Ø9A	B	Ø98	P	5
N	Ø1Ø ØAØ	B	Ø9E	C	3
I	Ø1Ø ØA3	B	ØA1	C	6
DRU	Ø1Ø ØFØ	B	ØEE	C	16
J	Ø1Ø 168	B	166	P	2
A	Ø1Ø 16A	B	168	P	5
KN	Ø1Ø 16F	B	16D	P	2
R	Ø1Ø 171	B	16F	P	5
MULT	Ø1Ø 176	B	174	C	1Ø

Anmerkungen:
1) Zur Sicherstellung der Eingabe 80spaltiger Lochkarten sind im Anschluß an I 71 Bytes freigelassen worden.
2) Da nach Voraussetzung eine Druckzeile 120 Stellen umfassen soll, sind im Anschluß an DRU 104 Stellen freigelassen worden.

Abb. 20.6. Symboltabelle

Sedezimale Hauptspeicheradresse	*Instruktion*	*Bemerkung*
Ø1Ø ØØØ	Ø5 BØ	-
Ø1Ø ØØ2	READ Ø1ØØAØ,Ø1ØØ78	keine Instruktion
	F2 12 B 16D B Ø9E	-
	F2 45 B 16F B ØA1	-
	FA 44 B 16F B Ø98	-
	D2 Ø4 B 168 B Ø98	-
	D2 Ø1 B 166 B Ø96	-
Ø1Ø Ø3Ø	F8 94 B 174 B 168	-
	FC 94 B 174 B 16F	-
	D1 ØØ B 17A B 17D	-
	D2 Ø4 B 168 B 176	-
	D2 ØF B ØEE B Ø86	-
	DE ØF B ØEE B 166	-
	FA 11 B 166 B Ø96	-
	F9 11 B 166 B 16D	-
	47 2Ø B ØØØ	-
	PRINT Ø1ØØFØ	keine Instruktion
	47 FØ B Ø2E	-
Ø1Ø Ø78	EOJ	keine Instruktion

Abb. 20.7. Programm in der Maschinensprache

In die Symboltabelle haben wir auch den Datentyp mit aufgenommen: P für gepackte Dezimaldaten, C für Zeichendaten. Mit Ausnahme der Tatsache, daß die Felder für N und I eingabebedingt aneinanderstoßen müssen, haben wir die Symbole in einer willkürlichen Reihenfolge angeordnet.

Unter Verwendung der fertig zusammengestellten Symboltabelle sind wir jetzt in der Lage, unsere symbolischen Instruktionen in die Maschinensprache zu übersetzen. Dabei drücken wir jede Adresse der obigen Tabelle (mit Ausnahme von SØ) mit Hilfe des Basisregisters aus. Das Programm in der Maschinensprache finden wir in Abb. 20.7.

Wir können nunmehr auch die Spezifikationen unserer Konstanten vervollständigen (Abb. 20.8.).

Sedezimale Hauptspeicheradresse	*Konstante*
Ø1Ø Ø88	4Ø2Ø 2Ø2Ø 224Ø 2Ø2Ø 214B 2Ø2Ø 2Ø2Ø 2Ø2Ø
Ø1Ø Ø98	ØØ1C
Ø1Ø Ø9A	ØØ1Ø ØØØØ ØC

Abb. 20.8. Konstanten

20.3. Die Assembliersprache für das System IBM/370

Für das System IBM/370 sind Assemblierer-Versionen für das Betriebssystem DOS und für das Betriebssystem OS verfügbar. Sie unterscheiden sich nur unwesentlich voneinander. Die geringfügigen Differenzen behandeln wir dort, wo wir ihnen begegnen.

Programme in der AL werden im allgemeinen auf spezielle Programmschemen (siehe Abb. 20.9.) geschrieben. In die Spalten 1 bis 8 wird der Name eingetragen, soweit ein solcher der Instruktion, der Konstanten oder dem Gebiet gegeben wird, die auf der betreffenden Zeile beschrieben sind. Die Spalte 9 bleibt leer. Die Spalten 10 bis 14 enthalten den mnemonischen Operationsschlüssel, die Spalte 15 bleibt wiederum leer. In die Spalten 16 bis 71 werden die Operanden eingetragen, wobei diese untereinander durch Kommas getrennt werden. Nach dem letzten Operanden verbleibt ein Zwischenraum. Im Anschluß an den letzten Operanden können beliebig lange Bemerkungen folgen, Hauptsache sie sind von diesem durch mindestens eine Leerstelle getrennt. Spalte 72 ist die Folgespalte; in der Regel bleibt sie leer. Eine vom Leerzeichen verschiedene Notation in dieser Spalte zeigt jedoch an, daß das Statement auf der nächsten Zeile fortgesetzt ist. Wird von dieser Fortsetzungsmöglichkeit Gebrauch gemacht, so müssen die ersten 15 Spalten der nächsten Zeile leer bleiben; das Statement darf erst ab Spalte 16 weitergeschrieben werden. Die Spalten 73 bis 80 werden vom Assemblierer nicht gelesen. Meist wird in diese Spalten eine Kennzeichnung untergebracht, wobei im allgemeinen die Informationen auf den Spalten 73 bis 76 das Programm identifizieren und die Spalten 77 bis 80 zur fortlaufenden Numerierung der Karten benutzt werden.

Zwei Fortsetzungszeilen sind erlaubt. Bemerkungen können auf allen Zeilen untergebracht werden. Mit Ausnahme der letzten Zeile müssen dazu alle anderen Zeilen mit einem Operanden enden, dem sich ein Komma und mindestens ein Leerzeichen anschliessen; erst danach darf die Bemerkung niedergeschrieben werden. Bei der letzten Zeile ist genau so zu verfahren wie weiter oben für Einzelzeilen beschrieben ist, d.h. vor evtl. Bemerkungen muß mindestens eine Leerstelle verbleiben.

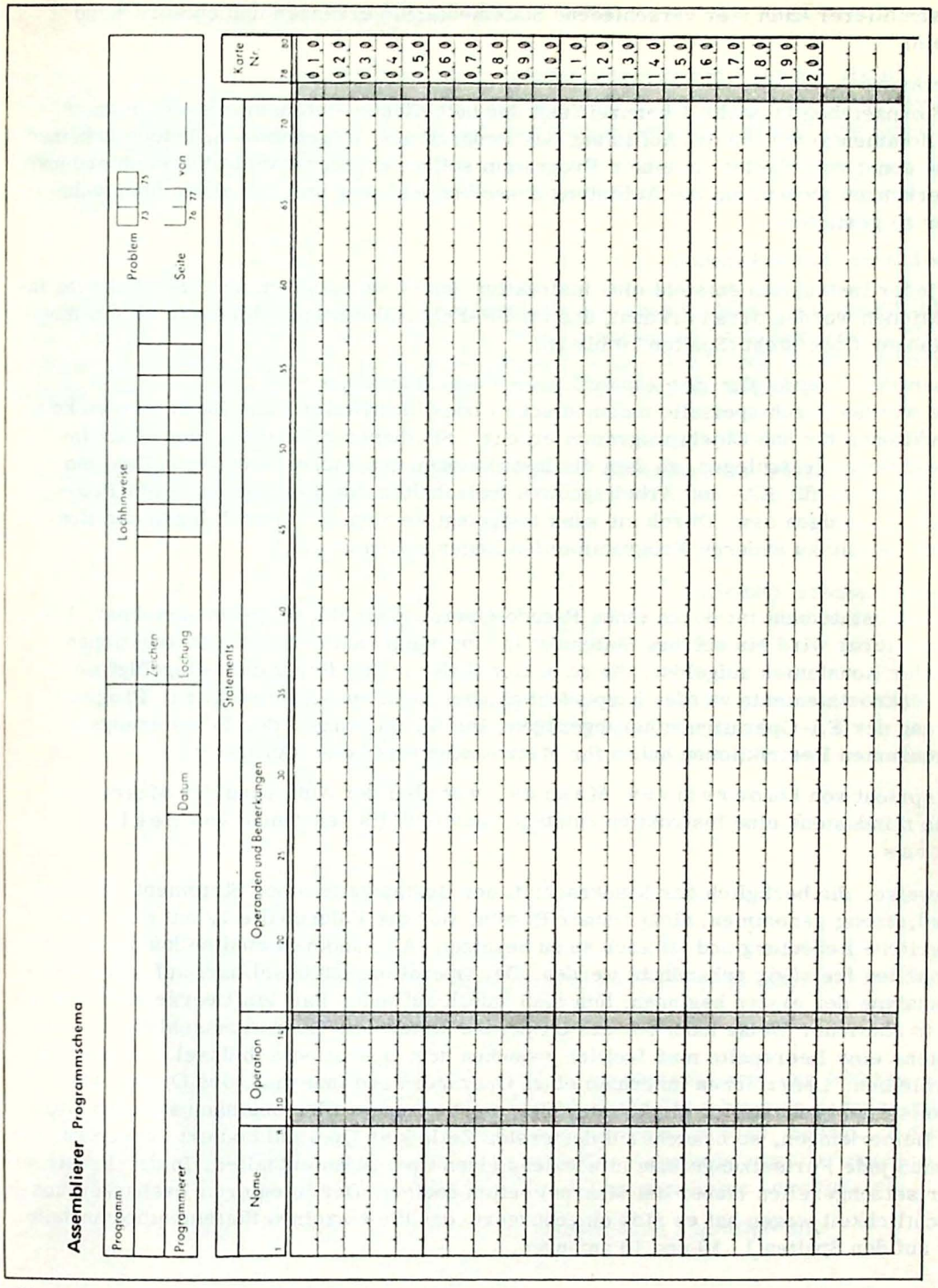


Abb. 20.9. Assembler-Programmschema[1)]

1) Bei der Niederschrift von Programmen hat es sich zur Vorbeugung gegen Verwechslungen eingebürgert, die Ziffer 0 vom Buchstaben O dadurch zu unterscheiden, daß die Null durchgestrichen, also wie Ø geschrieben wird.

Der Assemblierer kann vier verschiedene Statementarten erkennen und entsprechend behandeln:

1) *Kommentare*
   Ein Sternzeichen in Spalte 1 kennzeichnet das betreffende Statement als Kommentar. Ein Kommentar wird in die Auflistung des Programmes aufgenommen, jedoch in keiner Weise sonst verarbeitet. In einem Programm sollten möglichst viele Kommentare und Bemerkungen stehen, um die Auflistung desselben lesbarer und vor allem übersichtlicher zu gestalten.

2) *Symbolische Instruktionen*
   Aus jeder Instruktion entsteht eine Instruktion des Objektprogrammes. Symbolische Instruktionen werden daran erkannt, daß im OP-Feld (Operationsschlüsselfeld) ein mnemonischer Code steht (Spalten 10 bis 15).

3) *Steueranweisungen für den Assemblierer (Vereinbarungen)* [1]
   Diese werden durch spezielle mnemonische Codes identifiziert; aus ihnen werden keine Instruktionen für das Objektprogramm erzeugt. Sie dienen z.B. dazu, den Platz im Hauptspeicher festzulegen, ab dem die Instruktionen gespeichert werden sollen, um Speicherplätze für EA- und Arbeitsgebiete freizuhalten, um Konstanten in ein Programm einzufügen usw. Durch sie kann festgelegt werden, wo Instruktionen und Konstanten relativ zu anderen Programmteilen liegen sollen u.a.m.

4) *Makrostatements (Makros)* [2]
   Ein Makrostatement ist durch einen Pseudo-Operationsschlüssel gekennzeichnet. Vom Assemblierer wird ein solches Statement in eine Folge von mehreren Instruktionen und/oder Konstanten aufgelöst, die an seiner Stelle in das Programm eingefügt werden. Makrostatements werden hauptsächlich,aber nicht ausschließlich, zur Programmierung der EA-Operationen herangezogen. Die für die Anzahl der Fortsetzungszeilen genannten Restriktionen haben für Makrostatements keine Gültigkeit.

   Man spricht von imperativen Makros, wenn bei der Auflösung des Makrostatements mindestens eine Instruktion entsteht, anderenfalls redet man von deklarativen Makros.

Die Hinweise, die bezüglich der Niederschrift der Bestandteile eines Statements gegeben wurden, sind, streng genommen, nicht immer Regeln. Nur die Folgespalte 72 hat eine feste unveränderliche Bedeutung und ist auch so zu benutzen. Alle anderen Spalten können bezüglich ihres Inhaltes freizügig gehandhabt werden. Der Operationsschlüssel darf auf jeder Spalte mit Ausnahme der ersten beginnen; ihm muß jedoch auf jeden Fall ein Leerzeichen vorausgehen. In ähnlicher Weise kann der erste Operand hinter dem Operationsschlüssel starten; mindestens eine Leerspalte muß freilich zwischen dem Operationsschlüssel und dem Operanden verbleiben. Leerzeichen innerhalb eines Operanden und innerhalb des Operationsschlüssels sind grunsätzlich strikt untersagt. Fortsetzungszeilen beginnen stets ab Spalte 16. Fehlen Bemerkungen, so braucht auf der ersten Zeile kein Operand codiert zu werden. Hingegen muß jede Fortsetzungszeile mindestens einen Operanden enthalten. In der Praxis werden Fortsetzungszeilen (außer bei Makros) selten codiert. Der leichteren Lesbarkeit und der Übersichtlichkeit wegen hat es sich eingebürgert, daß die einzelnen Statementbestandteile immer auf den Spalten 1, 10 und 16 anfangen.

1) Auch als Assembliereranweisung bezeichnet.
2) Auch als Makrobefehle bezeichnet.

20.4. Fortsetzung des Beispiels

Wir wollen das Beispiel aus Abschnitt 20.2. wieder aufgreifen und sehen, wie es in der Assembliersprache (AL) [1] geschrieben werden muß. In späteren Abschnitten werden wir dann die Einzelheiten kennenlernen, die uns Auskunft darüber geben, wie jedes Statement geschrieben werden muß.

Wir beginnen mit den beiden Statements:

```
SØ          BALR  11,Ø
            USING *,11
```

Das USING-Statement ist eine Steueranweisung für den Assemblierer. Es besagt ihm, daß er annehmen kann, daß beim Ablauf des Objektprogrammes das Register 11 die Adresse des nächstfolgenden Hauptspeicherplatzes enthält (sie wurde durch die vorangegangene BALR-Instruktion geladen). Wir fahren fort mit dem Makro [2].

```
S1          READ  N,S3
```

Danach folgen die weiteren Instruktionen und eine Vereinbarung des Programmes; sie sind in Abb. 20.10. aufgeführt.

```
            PACK  KN,N
            PACK  R,I
            AP    R,KAP
            MVC   A,KAP
            MVC   J,EINS
S2          ZAP   MULT,A
            MP    MULT,R
            MVN   MULT+6(1),MULT+9
            MVC   A,MULT+2
            MVC   DRU,AUF
            ED    DRU,J
            AP    J,EINS
            CP    J,KN
            BC    2,S1
            PRINT DRU
            BC    15,S2
S3          EOJ
* KONSTANTEN UND VARIABLEN
AUF         DC    X'4ØØ2Ø2Ø2Ø22 4ØØ2Ø2Ø214B2Ø2Ø2Ø2Ø2Ø2Ø'
```

Abb. 20.10. Weitere Statements des Beispielprogrammes

In Abb. 20.10. finden wir nach Abschluß des Mixes aus Instruktionen und Makros einen Kommentar, der zu den Vereinbarungen überleitet. Durch DC wird eine Konstante ins Programm installiert. Das X, mit dem der Operand eingeleitet wird, besagt, daß die Konstante mit Sedezimalziffern definiert ist. Für diese Konstante wird der Speicherplatz unmittelbar nach dem letzten, von den Instruktionen belegten Bytes bereitgestellt.

Wir setzen die Codierung mit zwei weiteren Konstanten fort, nämlich mit

```
EINS        DC    PL2'1'
KAP         DC    PL5'1ØØØØØØ'
```

1) AL ist die Abkürzung von Assembler Language (Assembliersprache)

2) Makros für das Betriebssystem DOS behandeln wir im Kapitel 21, solche für das Betriebssystem OS im Kapitel 22

Das P zeigt bei ihnen an, daß es sich um Dezimalkonstanten in gepackter Darstellung handelt. Die auf L folgende Zahl nennt die Länge der Konstanten in Bytes.

Zur Reservierung von Speicherplatz,ohne in diesen zu Programmbeginn einen Wert (Konstante) hineinzustellen, bedienen wir uns der DS-Vereinbarung [1]. Auf so definierten Speicherplätzen kann im Programmverlauf ein Wert plaziert werden. Auf diese Weise richten wir also Variablen ein. Die Variablendefinitionen sind in die Abb. 20.11. eingeschlossen.

```
            DS    CL1
N           DS    CL3
I           DS    CL6
            DS    CL71
DRU         DS    CL16
            DC    CL1Ø4' '    LEERZEICHEN
J           DS    PL2
A           DS    PL5
KN          DS    PL2
R           DS    PL5
MULT        DS    CL1Ø
            END   SØ
```

Abb. 20.11. Fortsetzung der Vereinbarungen im Beispielprogramm

Das letzte Statement unseres Programmes ist schließlich

```
            END   SØ
```

Bei ihm handelt es sich um eine Steueranweisung für den Assemblierer. Das END-Statement muß stets als physisch letztes Statement eines Quellenprogrammes erscheinen. Der angegebene Operand verweist auf den Speicherplatz, zu dem vom Betriebssystem zuerst die Programmablaufsteuerung übertragen werden soll, wenn mit der Programmausführung begonnen wird. In diesem Fall ist es der Speicherplatz, auf dem die erste Instruktion (BALR-Instruktion) gespeichert ist, der wir bekanntlich das Symbol SØ gaben.

Es wäre abschließend vielleicht zu bemerken, daß wir Instruktionen und Speichergebiete, auf die wir uns im weiteren Programmverlauf nicht zu beziehen brauchen, auch nicht benannt haben. Neben größerer Übersichtlichkeit des Quellenprogrammes resultiert daraus auch eine geringfügige Leistungsverbesserung des Assemblierers bei der Übersetzung, was auf den Aufbau und die Verwaltung einer kleineren Symboltabelle zurückzuführen ist. Auf den Ablauf des Objektprogrammes hat ein solches Vorgehen keinen Einfluß.

20.5. Glieder (Terme) und Ausdrücke

Wie wir im letzten Abschnitt gesehen haben, darf ein Symbol als Name einer Instruktion oder eines Datengebietes verwendet werden. Symbole müssen aus mindestens einem Zeichen bestehen und dürfen höchstens acht Zeichen umfassen. Das erste Zeichen muß ein Alphabetzeichen sein, die anderen können Buchstaben oder Ziffern sein; Leerzeichen und andere Sonderzeichen dürfen in Symbole nicht aufgenommen werden. Die drei Zeichen $, # und @ werden von allen Assemblierern als Buchstaben angesehen. Dadurch ist es möglich, gewisse nichtenglische Alphabete zu benutzen. Einige gültige Symbole sollen jetzt als Beispiele angeführt werden:

1) DS ist die Abkürzung von „Define Storage" („Definieren Speicher")

```
UMIL56R
L
#
#345Ø999
IFABK35Ø
MARIANNE
```

Die folgenden Symbole sind ungültig:

4RT	(beginnt nicht mit einem Buchstaben)
TT+R8	(enthält ein Sonderzeichen)
ABCDEFGH	(enthält mehr als acht Zeichen)
EIN BER	(enthält ein Leerzeichen, ein Sonderzeichen also)

Man sagt, daß ein Symbol definiert ist, wenn es im Namensfeld eines Statements erscheint. Jedes in einem Programm verwendete Symbol muß irgendwo definiert sein. Kein Symbol darf in einem Programm mehr als einmal definiert werden. Ausnahmen gelten nur für die Namen von Programmabschnitten. Doch darüber sprechen wir später.

Unter einem sich selbst definierenden Glied [1] verstehen wir ein Glied, dessen Wert in ihm selbst enthalten ist. Ein solches Glied wollen wir auch als Direktwert bezeichnen.

Unter dezimalen Direktwerten versteht man ganze Dezimalzahlen ohne Vorzeichen wie 9, 73, 0, 4096 usw. Innerhalb der Zahlen dürfen keine Kommas oder Punkte stehen. Die Dezimalzahl wird vom Assemblierer in die entsprechende Dualzahl konvertiert.

Unter sedezimalen Direktwerten versteht man Glieder der Form X'ss···s', wobei unter s Sedezimalziffern zu verstehen sind. Verfolgen wir einige Beispiele hierfür: X'8', X'4E56DF', X'FFF6', X'1F'. Je zwei Sedezimalziffern belegen ein Byte.

Duale Direktwerte besitzen die Form B'dd···d', wobei unter d Dualziffern oder Bits zu verstehen sind, d.h. 0 oder 1. So wird z.B. durch B'00101110' eine aus acht Dualziffern bestehende Dualzahl definiert, die also gerade acht Bitstellen, d.h. ein Byte belegt. Statt des dualen Direktwertes hätte man hier auch einen sedezimalen Direktwert, hier X'2E', niederschreiben können; am Inhalt des Bytes hätte sich nichts geändert.

Falls erforderlich, werden bei dezimalen, sedezimalen und dualen Direktwerten nach links Nullen eingefügt, führende Nullen also.

Zeichendirektwerte besitzen die Form C'zz· · ·z'; unter den z werden hierbei beliebige Zeichen verstanden. Außer den beiden Zeichen ' und & dürfen alle anderen Zeichen ohne weiteres zwischen die Anführungszeichen gestellt werden. Um ein ' oder ein & zu spezifizieren, muß das gewünschte Zeichen zweimal hintereinander niedergeschrieben werden. Durch C'9''6' wird somit ein drei Bytes langes Feld mit dem Inhalt 9'6 festgelegt, durch C'Y&&=' ebenfalls ein drei Byte langes Feld mit dem Inhalt Y&=. In einem Zeichendirektwert darf jedes Zeichen auftreten, das im EBCDI-Code verschlüsselbar ist.

Direktwerte werden zur Spezifizierung von Adressen, Längen, Registernummern usw. verwendet.

Unter einem Literal versteht man einfach eine Konstante, der ein Gleichheitszeichen (=) vorangeht. Somit sind z.B. =X'F1', =C'9X' und =B'011' Literale. Durch Literale

---

1) Statt Glied oder Term ist auch der Begriff einfacher Ausdruck geläufig.

wird dem Assemblierer mitgeteilt, daß er eine Konstante erzeugen soll, deren Wert gleich dem im Literal stehenden Wert ist und daß er die Adresse dieser Konstanten in dem Operanden verwenden soll, in dem das Literal erscheint.

Unter einem Glied oder Term versteht man ein Symbol oder ein Literal oder einen Direktwert. Unter einem Ausdruck versteht man eine Kombination von Gliedern, wobei jeweils zwei aufeinanderfolgende Glieder durch einen arithmetischen Operator verbunden sind. Folgende Operatoren sind erlaubt:

+	*Addition*
-	*Subtraktion*
*	*Multiplikation*
/	*Division*

Zusätzlich dürfen in Ausdrücken (runde) Klammern benutzt werden, wodurch man den Vorrang der arithmetischen Operationen entsprechend der algebraischen Regeln beeinflussen kann. Die Vorzeichen + und - dürfen ebenfalls in Ausdrücke eingehen. Maximal sind 19 Operatoren und 6 Klammerstufen erlaubt [1]. Der einem Glied zukommende Wert muß in ein Vollwort hineinpassen. - Statt Glied oder Term sagt man auch einfacher Ausdruck, verbundene einfache Ausdrücke heißen dann zusammengesetzte Ausdrücke; der Begriff „Ausdruck" schlechthin kennzeichnet dann sowohl Glieder als auch zusammengesetzte Ausdrücke.

Bei der Bildung von Ausdrücken sind weiterhin folgende Regeln zu beachten:

1) Literale sind in zusammengesetzten Ausdrücken nicht erlaubt. Somit darf z.B.

```
L       8,=X'ØØ14FØØ8'
```

geschrieben werden,

```
L       8,=X'ØØ14FØØ8'+14
```

jedoch nicht.

2) Bei der Division wird für die Weiterrechnung nur der Quotient verwendet, der Rest geht verloren; dabei erfolgt keine Rundung. Bei einer Division durch 0 wird das Resultat auf 0 gesetzt und kein Fehler angezeigt.

Ein Ausdruck heißt absolut, wenn sich sein Wert bei einer Programmverschiebung nicht ändert; er heißt verschieblich, wenn sich sein Wert bei einer Programmverschiebung ändert. Machen wir uns diesen Sachverhalt doch einmal am Beispiel des vorigen Abschnittes 20.4. klar, indem wir einige dieses Programm betreffende Ausdrücke bilden. Gemäß der dort eingeführten Symbole ist S3-S2 ein absoluter Ausdruck, d.h. er besitzt ständig den gleichen Wert, nämlich 72 (dezimal) bzw. 48 (sedezimal). Der Ausdruck S2+3Ø ist jedoch ein verschieblicher Ausdruck, weil bei einer Verschiebung des Programmes auf andere Hauptspeicherplätze S2 seinen Wert ändert, damit ändert aber auch die Summe S2+3Ø ihren Wert. Wenn ein Programm verschoben wird, so ändern sowohl S2 als auch S3 ihre Werte um den gleichen Betrag; somit bleibt ihre Differenz konstant.

Multiplikationen und Divisionen von verschieblichen Ausdrücken sind nicht erlaubt. Somit darf S3/S2 nicht codiert werden. Gestattet ist jedoch die Niederschrift von (S3-S2)/3, da S3-S2 ein absoluter Ausdruck ist.

---

1) Für den Assemblierer des Betriebssystemes DOS sind maximal nur 15 Operatoren und 5 Klammerstufen erlaubt. Außerdem darf der Wert eines Gliedes nicht größer sein, als ein Zahlenwert, den drei Bytes aufnehmen können.

Müssen in einem absoluten Ausdruck verschiebliche einfache Ausdrücke enthalten sein, so müssen diese paarweise so auftreten, daß sie die Folgen einer Verschiebung aufheben.

Unter einem komplexen verschieblichen Ausdruck versteht man einen verschieblichen Ausdruck, bei dem nach dem Streichen aller möglichen Paare von verschieblichen Ausdrücken das Ergebnis in sich entweder einen negativen verschieblichen Ausdruck oder eine Summe aus mehreren verschieblichen Ausdrücken einschließt. Damit ist wohl klar, daß S3-S2+1 kein komplexer verschieblicher Ausdruck ist, S3-S2-S1 und S3+S2 sind dagegen solche.

Verschiebliche einfache Ausdrücke dürfen nicht in die Operationen „Multiplizieren" und „Dividieren" eingehen. Verschiebliche Ausdrücke enthalten immer eine ungerade Anzahl verschieblicher einfacher Ausdrücke.

Um Bezugnahmen auf die Adresse der laufenden Instruktion zu erleichtern, kann ein Sternzeichen als erster einfacher Ausdruck in einem zusammengesetzten Ausdruck geschrieben werden. Das Sternzeichen repräsentiert dabei die Adresse des ersten Bytes der Instruktion. Zwischen dem Sternzeichen, das eine Multiplikation anzeigt, und dem für den Adreßzähler [1] verwendeten Sternzeichen kann es zu keiner Verwechslung kommen. Beispiele: Bei *+7, *+BK-BL und *-4 kennzeichnet das Sternzeichen den Adreßzähler, bei 7 *(BK-BL) kennzeichnet es eine Multiplikation. Das Sternzeichen kann in dieser Bedeutung auch als einziges alleiniges Glied eines Ausdruckes auftreten.

Beim Schreiben von Programmen in der AL besteht der einzige Zweck von Ausdrücken darin, daß mit ihrer Hilfe Adressen berechnet werden können. Meistens können die Adressen durch recht einfache Ausdrücke angegeben werden; erinnern wir uns, daß die kompliziertesten Ausdrücke bei unserem im letzten Abschnitt behandelten Beispiel die Formen MULT+6, MULT+9 und MULT+2 hatten.

20.6. Die USING-Vereinbarung

Als erste Instruktion jedes Programmes in der AL wird gewöhnlich

```
BALR  r,Ø
```

verwendet. Hierbei bedeutet r das Register, das als Basisregister verwendet wird. Beim Schreiben eines Programmes sind die durch den Programmierer angegebenen und benutzten Adressen relativ zu einer Basisadresse. Ein längeres Programm erfordert die Verwendung von zwei oder mehr als zwei Basisregistern, da Distanzadressen [2] bekanntlich den Wert 4096 nicht überschreiten können. Der Assemblierer selbst weiß aber nicht, welche Basisregister verwendet werden oder welchen Wert sie enthalten dürfen, da die BALR-Instruktion weitaus mehr dazu benutzt wird, Unterprogramme miteinander zu verbinden [3], als Basisregister mit Werten zu versorgen. Der Programmierer kann, wenn er sich dazu entscheidet, für jeden Operanden das Basisregister extra angeben, doch wächst dadurch die Schreibarbeit beträchtlich an. Die Vereinbarung

```
USING  v,r1,···,rn
```

wird dazu benutzt, um dem Assemblierer mitzuteilen, daß das allgemeine Register r_1 den Wert v enthält. Hierbei bedeutet v einen absoluten oder verschieblichen Ausdruck. In den anderen spezifizierten Registern werden dann folgende Werte erwartet:

1) Auch als Zuordnungszähler bezeichnet.
2) Auch als relative Adressen bezeichnet.
3) Speicherung der Rückkehradresse und Ansprung des Unterprogrammes.

r_2	v+4096
r_3	v+8192 [=v+(3-1)·4096]
⋮	⋮
r_n	v+(n-1)·4096

Wird in einem USING-Statement das Register 0 als Operand aufgeführt, so muß es im ersten Operanden erscheinen; außerdem muß v ein verschieblicher Ausdruck sein. Der Assemblierer nimmt dann an, daß das Register 0 den Wert 0, das Register r_2 den Wert 4096, r_3 den Wert 8192 usw. enthält. Es ist dabei völlig gleichgültig, welcher Wert durch v repräsentiert wird.

In dem Beispiel, das wir im Abschnitt 20.4. behandelt haben, sagt die Vereinbarung

```
        USING  *,11
```

dem Assemblierer, daß der laufende Wert des Adreßzählers, d.h. die Adresse des nächsten noch nicht belegten Speicherplatzes, im Register 11 steht und daß das Register 11 als Basisregister verwendet werden kann.

Wird dem Assemblierer mehr als ein Basisregister zur Verfügung gestellt, so zieht der Assemblierer immer dasjenige zur Bildung der Adressen heran, das zur kleinsten Distanzadresse führt. Somit bewirken die Statements

```
        BALR   11,Ø              ADRESSE VON A INS REG.11
        USING  *,11,1Ø
        LR     1Ø,11             SPEICHERPLATZ A
        AH     1Ø,=H'4Ø96'       SPEICHERPLATZ A+2
```

folgendes: Wenn beispielsweise Register 11 durch die BALR-Instruktion mit 010002 (Adresse des Speicherplatzes A) geladen wird, so wird durch die Instruktion LR und AH ins Register der sedezimale Wert 010002 + 1000 = 011002 gestellt. Diesen Wert erwartet aber der Assemblierer aufgrund der getroffenen USING-Vereinbarung. Wenn die beiden nächsten Instruktionen die Instruktionen

```
        C      ABC,DEF           SPEICHERPLATZ A+6
        BC     8,*+8             SPEICHERPLATZ A+1Ø
```

sind, so wird zur Bildung der Verzweigungsadresse bei der BC-Instruktion das Register 11 herangezogen. Würde jedoch die BC-Instruktion wie folgt lauten:

```
        BC     8,*+4Ø94          SPEICHERPLATZ A+1Ø
```

so würde zur Adreßbildung das Register 10 benutzt werden, denn *+4094 ergibt dezimal dann 65548+4094=69642. Die höchste Hauptspeicheradresse jedoch, die unter Zuhilfenahme von Register 11 gebildet werden kann, ist nur gleich 65536+4095=69631, ist also kleiner als 69642. Im allgemeinen werden sich bei Benutzung mehrerer Basisregister die in ihnen stehenden Werte um 4096 unterscheiden. Diese Differenz muß nicht unbedingt vorliegen, wie es das nachfolgende Beispiel zeigt:

```
        BALR   11,Ø
        USING  *,11
A1      LA     9,4ØØØ
        AR     9,11
        USING  4ØØØ+A1,9
```

Durch diese Instruktionsfolge wird ins Register 11,sagen wir beispielsweise,der Wert 8194 und ins Register 9 der Wert 12194 gestellt. Da der Assemblierer zur Bildung der in den Instruktionen auftretenden Adressen immer das Basisregister heranzieht, dessen Inhalt die kleinsten Distanzadressen ermöglicht, benutzt der Assemblierer bei der Zerlegung von Hauptspeicheradressen zwischen 12194 und 12289 immer das Register 9 und nicht mehr das Register 11 wie bei den Adressen zwischen 8194 und 12193.

Register können für den weiteren Gebrauch auch gesperrt (aufgegeben) werden, indem das Statement

DROP $r_1,\cdots,r_n$

niedergeschrieben wird. Sollen aufgegebene Register erneut zur Benutzung für den Assemblierer verfügbar gemacht werden, so müssen sie in einem neuen USING-Statement erscheinen.

Es ist wichtig, die Tatsache festzuhalten, daß durch die USING-Vereinbarung kein Wert in irgendein Register gestellt wird. Dieses Statement informiert vielmehr den Assemblierer, daß für die Ausführung des Objektprogrammes Instruktionen erzeugt werden sollen, die die bei USING angegebenen Register als Basisregister verwenden. Dem Programmierer bleibt es überlassen, in sein Programm Instruktionen aufzunehmen, die die Register mit den erforderlichen Werten versorgen.

20.7. Instruktionen

Alle Maschineninstruktionen des RR-Formats besitzen bekanntlich die Form

OP R_1,R_2

Eine Ausnahme hiervon bilden lediglich die Instruktionen SVC und SPM. In der AL werden diese Instruktionen wie folgt geschrieben:

name OP r_1,r_2

Hierbei bedeuten r_1 und r_2 absolute Ausdrücke, deren Werte zwischen 0 und 15 (einschließlich)liegen dürfen. Der Assemblierer prüft, ob unkorrekte Angaben vorliegen, wie die Angabe eines Gleitpunktregisters mit ungerader Nummer oder die Angabe eines Registers mit ungerader Nummer für r_1 bei der Multiplikation. Wird ein Fehler entdeckt, so wird zusammen mit der Auflistung des Programmes eine Fehlernachricht gedruckt. Da bei der SPM-Instruktion R_2 nicht benutzt wird, wird diese Instruktion einfach wie folgt codiert:

name SPM r_1

Für die SVC-Instruktion gilt:

name SVC i

Hierbei ist i ein absoluter Ausdruck mit Werten zwischen 0 und 255.

Alle Instruktionen im RX-Format sind bekanntlich in der Maschinensprache wie folgt aufgebaut:

OP R_1,X_2,B_2,D_2

In der AL werden Instruktionen dieses Formats in der Form

name OP $r_1,D_2(X_2,B_2)$

geschrieben [1]. Hierbei stellt r_1 einen absoluten Ausdruck dar, während durch D_2, X_2 und B_2 der zweite Operand spezifiziert wird. D_2 muß ebenfalls ein absoluter Ausdruck sein. Wenn D_2=0 ist, so muß die 0 niedergeschrieben werden, da ein Statement in der Form

```
name    OP    r1,(X2,B2)
```

durch den Assemblierer als fehlerhaft gekennzeichnet wird. Wenn X_2 gleich 0 ist, dann muß ebenfalls für X_2 eine 0 geschrieben werden. Sind sowohl X_2 als auch B_2 gleich 0, dann darf das betreffende Statement in der Form

```
name    OP    r1,D2
```

codiert werden.

Ist für den Assemblierer mittels des USING-Statements ein Basisregister zur Verfügung gestellt worden, dann hat der Programmierer zusätzlich die Möglichkeit, ihm eine in zwei Teile aufgeteilte Adresse zu übergeben, nämlich die Teile D_2-B_2 und X_2. In einem solchen Fall kann die betreffende Instruktion in der Form

```
name    OP    r1,S2,(X2)
```

geschrieben werden. Wenn X_2=0 ist, dann darf diese Form auf

```
name    OP    r1,S2
```

verkürzt werden.

Der Assemblierer berechnet den Wert des Ausdrucks S_2 [2] und zerlegt diesen Wert so, daß er durch den Inhalt eines Basisregisters und einer Distanzadresse ausgedrückt werden kann. Es kommt hierbei zum Ausdrucken einer Fehlernachricht, wenn die Distanzadresse nicht mit Hilfe eines in einem USING-Statement spezifizierten Registers ausgedrückt werden kann.

Die Maschineninstruktionen BXH, BXLE. LM und STM liegen bekanntlich im RS-Format vor und besitzen die Form

```
OP      R1,R3,B2,D2
```

Diese Instruktion müssen wir in der AL wie folgt codieren:

```
name    OP    r1,r3,D2(B2)
```

oder

```
name    OP    r1,r3,S2
```

Die Verschiebungsinstruktionen verwenden R_3 nicht; deshalb können sie in einer der beiden nachstehenden Formen niedergeschrieben werden:

```
name    OP    r1,D2(B2)
```

bzw.

```
name    OP    r1,S2
```

Instruktionen im SI-Format treten in der Form

```
OP      I2,B1,D1
```

auf; in der AL sind sie jedoch wie folgt zu codieren:

```
name    OP    D1(B1),I2
```

bzw.

```
name    OP    S1,I2
```

1) Adressen, die durch D, B, X spezifiziert sind, werden als explizite Adressen bezeichnet.
2) S_2 wird als implizite Adresse bezeichnet.

Hierzu sollte bemerkt werden, daß die Operanden in der umgekehrten Reihenfolge geschrieben werden müssen, nicht also in der, in der sie in den Maschineninstruktionen erscheinen. Bei den Instruktionen mit den Operationsschlüsseln LPSW, SSM, TIO, TCH und TS wird das Direktbyte nicht verwendet, so daß für sie folgende Form in der AL zum Zuge kommt:

name	OP	$D_1,(B_1)$

bzw.

name	OP	S_1

Instruktionen im S-Format besitzen die gleiche Form wie die Instruktionen im SI-Format ohne Direktdaten.

Instruktionen im SS-Format, bei denen die Angabe von zwei Längen erforderlich ist, sind bekanntlich in der Maschinensprache wie folgt aufgebaut:

OP L_1,L_2,B_1,D_1,B_2,D_2

In der AL nehmen diese Instruktionen eine der folgenden Formen an:

name	OP	$D_1(L_1,B_1),D_2(L_2,B_2)$
name	OP	$D_1(L_1,B_1),S_2(L_2)$
name	OP	$S_1(L_1),D_2(L_2,B_2)$
name	OP	$S_1(L_1),S_2(L_2)$

Wenn die Länge eines Operanden vom symbolischen Namen her genommen werden soll, dann darf die Längenspezifikation weggelassen werden, so daß die Operandenangabe auf die Formen

D(,B)

oder

S

reduziert werden können. In der Maschinensprache reichen die Längenangaben von 0 bis 15 einschließlich und sind um 1 geringer als die Feldlänge. Bei den in der AL niedergeschriebenen Instruktionen gleichen die Längenangaben der wahren Länge (1 bis 16) der Felder. Der Assemblierer subtrahiert von der im Statement angegebenen Länge 1 und erhält auf diese Weise die für die Maschineninstruktionen erforderliche Länge.

Für jene Instruktionen im SS-Format, die nur eine Längenangabe (Wert von 0 bis 255) enthalten dürfen, nimmt die Maschineninstruktion bekanntlich die folgende Form an:

OP L,B_1,D_1,B_2,D_2

In der AL wird diese eine Längenangabe als Teil des ersten Operanden niedergeschrieben, so daß also die Instruktionen folgendes Aussehen bekommen:

name	OP	$D_1(L,B_1),D_2(B_2)$
name	OP	$S_1(L),D_2(B_2)$
name	OP	$D_1(L,B_1),S_2$
name	OP	$S_1(L),S_2$

Wenn der Längenoperand weggelassen wird, so nehmen die Instruktionen folgende vier Formen an:

name	OP	$D_1(,B_1),D_s(B_2)$
name	OP	$S_1,D_2(B_2)$
name	OP	$D_1(L,B_1),S_2$
name	OP	S_1,S_2

Bei diesen Formen wird vom Assemblierer diejenige Länge genommen, die dem ersten Operanden zugeordnet ist. Man spricht hierbei von einer impliziten Länge.

Für die SRP [1]-Instruktion besitzt das Maschinenformat bekanntlich die Form

OP L_1,I_3,B_1,D_1,B_2,D_2

In der AL ist diese Instruktion in einer der nachstehenden Formen zu codieren:

name	SRP	$D_1(L_1,B_1),D_2(B_2),I_3$
name	SRP	$S_1(L_1),S_2,I_3$
name	SRP	S_1,S_2,I_3

usw.

Wenn in einem Operanden ein Ausdruck erscheint, dann gilt als implizite Länge dieses Ausdruckes die dem ersten Symbol zugeordnete Länge. Wenn also z.B. XY8 als Symbol mit einer Länge von acht Bytes definiert wurde und XY5 mit einer Länge von fünf Bytes, dann wird der Längenspezifikation der Instruktion

```
MVC   XY5,XY8
```

implizit eine Länge von fünf Bytes zugewiesen, d.h. die in die Maschinensprache übersetzte Instruktion besitzt folgende Struktur:

```
D2 Ø4 ···
```

Bei der Instruktion

```
MVC   XY5-XY8+9ØØØ,XY8
```

ergibt sich ebenfalls implizit eine Längenangabe von 5 Bytes, da XY5 das erste Symbol in dem Ausdruck ist, der für den ersten Operanden niedergeschrieben wurde.

Im Anhang J befindet sich eine Übersicht über die Operandenformen, die bei Instruktionen in der AL verwendet werden dürfen.

20.8. Erweiterte mnemonische Operationsschlüssel

Da Programmierer die BC- und BCR-Instruktionen häufig niederschreiben müssen, sind erweiterte Operationsschlüssel geschaffen worden, so daß sich der Programmierer nicht daran zu erinnern braucht, welche Maskenbits bei einer bestimmten Verzweigungsbedingung gesetzt sein müssen. Für beide Instruktionen sind insgesamt je 19 erweiterte Operationsschlüssel vorgesehen; diese sind komplett im Anhang J aufgeführt.

Zwei von diesen erweiterten Operationsschlüsseln sind nicht an eine Bedingung geknüpft. Sie lauten für die BC-Instruktion

```
B     a          und
NOP   a
```

1) Shift and Round decimal

und sind äquivalent den Instruktionen

```
B       15,a        und
B       Ø,a
```

Hierbei bedeutet a einen Ausdruck.

Für die Instruktion BCR lauten die entsprechenden Formen

```
BR      r2          und
NOPR    r2
```

Beide sind äquivalent den bisher benutzten Formen

```
BCR     15,r2  ——►  unbedingte Verzweigung
BCR     Ø,r2   ——►  keine Operation (Nop)
```

Wenn der erweiterte Operationsschlüssel niedergeschrieben wird, so wird nur ein Operand notiert, und zwar derjenige, der normalerweise als zweiter erscheint.

Für bedingte Verzweigungen sind die restlichen 17 erweiterten Operationsschlüssel geschaffen worden. Diese schließen bei der BC-Instruktion u.a. ein:

```
BH
BNZ
BE
BNM
```

Die aufgeführten vier sind äquivalent den folgenden Notierungen:

```
BC      2,···
BC      7,···
BC      8,···
BC      11,···
```

Sie bedeuten der Reihe nach eine Verzweigung bei folgenden Bedingungen: hoch (engl: high), nicht Null (engl. not zero), gleich (engl. equal), nicht minus (engl. not minus).

Aufgabengruppe 54

1) Welcher Wert wird unter Y gespeichert, wenn das folgende Programm abgelaufen ist?

```
S       BALR    11,Ø
        USING   *,11
        L       7,X
        M       6,X+4
        SR      6,7
        ST      6,Y
        EOJ
X       DC      X'ØØØØØØØ3'
        DC      X'ØØØØØØØ7'
Y       DS      CL4
        END     S
```

2) Welches Ergebnis zeitigt das folgende Programm, wenn es auf einer IBM /370 abläuft, deren Hauptspeicherkapazität 65 536 Bytes beträgt?

```
G       BALR  11,Ø
        USING *,11
        LM    6,9,W
        SR    7,6
H       ST    9,Ø(7)
        BXH   8,6,H
        EOJ
W       DC    X'FFFFFFFC'
        DC    X'ØØØ1ØØØØ'
        DC    X'ØØØØ4ØØØ'
        DC    X'ØØØØØØØØ'
        END   G
```

3) Gegeben ist

```
X       DS    F
Y       DS    F
```

Schreibe eine Instruktionsfolge, die in Y den Wert von $3X^2-7X+4$ stellt. Vorausgesetzt ist, daß $|X| \leq 3000$ ist. Hierbei wird durch F eine Konstante angezeigt, die ein volles Wort belegt, bzw. ein ebenso großes Gebiet freigehalten.

4) Schreibe ein Unterprogramm, dem eine positive ganze Zahl im Register 1 als Argument übergeben wird. Im Unterprogramm ist die Quadratwurzel aus dem Argument zu berechnen und das Ergebnis ins Register 0 zu stellen. Das Unterprogramm soll mit einer Sprunginstruktion beendet werden, die eine Verzweigung zu der im Register 14 stehenden Adresse bewirkt. Zur Bestimmung der Quadratwurzel ist die Formel zu verwenden, die mit Hilfe des Newtonschen Näherungsverfahrens abgeleitet werden kann. Sie lautet:

$$x_{n+1} = \frac{1}{2}\left(x_n - \frac{a}{x_n}\right)$$

Der Radikand ist hierbei mit a bezeichnet worden; als erster Näherungswert ist $x_0=1$ zu wählen. Zuerst berechnen wir mit Hilfe der Formel x_1, danach verwenden wir x_1, um x_2 zu bestimmen usw. Wir setzen dieses Iterationsverfahren so lange fort, bis wir zwei aufeinanderfolgende Wurzeln gefunden haben, die innerhalb der erlaubten Fehlergrenze gleich sind. In unserem Fall soll das Ergebnis bis auf 0, 1 genau berechnet werden.

5) Ändere Aufgabe 4) wie folgt ab: Ins Register 0 soll eine 0 gestellt werden, wenn der Radikand nicht positiv ist. Rette außerdem die Inhalte aller Register und speichere in sie am Schluß des Unterprogrammes die Werte zurück, die sie beim Aufruf des Unterprogrammes aufwiesen. Als Basisregister soll Register 15 verwendet werden.

20.9. Zeichenkonstanten

Ins Objektprogramm werden Konstanten eingefügt, wenn man das DC-Statement [1] verwendet. Seine allgemeine Form lautet:

```
name      DC    operand
```

Wird ein Name niedergeschrieben, so wird diesem ein Wert zugeordnet, der gleich der Adresse des ersten Bytes der Konstanten ist. Wenn in einem Statement mehr als eine Konstante definiert wird, so bezieht sich der Name auf die erste Konstante. Dem Namen wird eine implizite Länge zugeordnet, die gleich der Länge der ersten Konstanten ist.

Am einfachsten können Konstanten vom Zeichentyp definiert werden. Man braucht sie nur im Operanden wie folgt zu codieren:

```
C'zeichen'
```

Durch diese Definition wird ein Feld reserviert, dessen Länge durch die in der Konstanten enthaltenen Zeichen bestimmt ist. Beispiel:

```
C'BEISPIEL NR. 1'
```

Hierdurch wird für die Konstante ein Feld reserviert, das eine Länge von 14 Bytes besitzt. In dieses Feld wird die Konstante hineingestellt; sie gehört damit zum Objektprogramm. Der Konstanten geht der Buchstabe C voraus, der sie als zugehörig zum Zeichentyp charakterisiert. Die Konstante selbst wird in Anführungszeichen eingeschlossen. Soll ein Anführungszeichen (') oder ein Undzeichen (&) in eine Konstante aufgenommen werden, so muß das betreffende Zeichen zweimal niedergeschrieben werden. Es wird aber selbstverständlich nur einmal gezählt. Somit definieren

```
C'A&&R CO.'  bzw.
C'WENN ''S'
```

die beiden Konstanten A&R CO. bzw. WENN'S; ihre Längen betragen 7 bzw. 6 Bytes.

Konstanten vom C-Typ können auch in der Form

```
CLn'zeichen'
```

codiert werden. Hierbei wird durch n die gewünschte Länge festgelegt. Überschreitet die Zahl der niedergeschriebenen Zeichen n, dann werden in die Konstante nur die ersten n Zeichen aufgenommen. Ist die Anzahl der niedergeschriebenen Zeichen dagegen kleiner als n, so wird die Konstante vom Assembler nach rechts mit Leerzeichen bis zur spezifizierten Länge aufgefüllt. Somit wird durch

```
C'A&&R CO.'
```

eine Konstante mit 7 Bytes Länge definiert, durch

```
CL9'A&&R CO.'
```

---

1) DC - Define Constant (Definiere Konstante)

dagegen eine Konstante von 9 Bytes Länge, nämlich A&RbCO.bb [1]. Gibt man

```
CL5'A&&R CO.'
```

an, so wird die Konstante A&RbC erzeugt, die 5 Bytes lang ist [1].

Dem Buchstaben C darf ferner ein Wiederholungsfaktor vorausgehen. Mit diesem nimmt der Operand folgende Form an:

```
dCLn'zeichen'
```

Der Wiederholungsfaktor d bewirkt, daß d identische Konstanten vom Assemblierer erzeugt und in aufeinanderfolgende Speicherplätze gespeichert werden. Wird auf die Angabe von d verzichtet, so wird es zu 1 angenommen. Wenn d gleich Null ist, so wird keine Konstante erzeugt und auch kein Speicherplatz reserviert. Allerdings kann damit einem solchen Statement ein Symbol und eine Länge zugeordnet werden. Sehen wir uns das an einem Beispiel an:

```
HAAR      DC     ØC'ABC'
```

Bei diesem Beispiel wird die Zeichenkonstante ABC nicht Teil des Objektprogrammes, doch dem Symbol HAAR wird die Adresse des Hauptspeicherplatzes zugewiesen, der das nächste Byte darstellt. HAAR bekommt außerdem die implizite Länge 3. Ein auf diese Weise in ein Programm eingeführtes Symbol kann in Ausdrücken verwendet werden, die zu anderen Statements gehören.

Ein Wiederholungsfaktor erzielt den gleichen Effekt, der durch zwei- oder mehrmaliges Schreiben der gleichen Konstanten eintritt. Somit sind die beiden Definitionen

```
DC     C'DEF',C'DEF',C'DEF'
```

und

```
DC     3C'DEF'
```

einander gleichwertig. In beiden Fällen ergibt sich die folgende Hauptspeicherbelegung

```
DEFDEFDEF
```

Die maximale Länge einer Zeichenkonstanten beträgt 256 Bytes.

Zwei weitere Beispiele von Zeichenkonstanten sollen die Betrachtungen über diesen Konstantentyp abschließen. Durch

```
DC     3CL4'ABCDEF'
```

werden beispielsweise drei Konstanten definiert, die jeweils eine Länge von vier Bytes besitzen. Im Hauptspeicher steht dann

```
ABCDABCDABCD
```

Betrachten wir das zweite Beispiel. Durch

```
DC     3C'ABCDEF'
```

werden ebenfalls drei Konstanten definiert, die jeweils sechs Bytes umfassen; im Hauptspeicher kommt es zu folgender Speicherbelegung:

```
ABCDEFABCDEFABCDEF
```

---

1) b bedeutet hier das Leerzeichen.

Die Länge einer Konstanten kann statt in Bytes auch durch die Anzahl der von ihr zu belegenden Bitstellen angegeben werden. Der Längenoperand besitzt dann die Form

```
L.n
```

Hierbei kann n den Wertebereich von n=1 bis n=2048 abdecken. Sind Längenangaben von Konstanten nicht durch Nennung der Bitstellenzahl getätigt, so beginnen die Konstanten grundsätzlich an einer Bytegrenze. Bei einer Längenspezifikation in Bitstellen beginnt die erste Konstante eines Statements ebenfalls an einer Bytegrenze, die folgenden jedoch auf der nächsten verfügbaren Bitstelle. Nach Abschluß der letzten Konstanten werden die verbleibenden Bitstellen des letzten Bytes auf Null gesetzt. Betrachten wir hierzu ein Beispiel:

```
        DC      CL.13'AC',2CL.5'3',CL1'9'
```

Aus dieser Vereinbarung wird vom Assemblierer die Hauptspeicherbelegung generiert, die in Abb. 20.12. dargestellt ist.

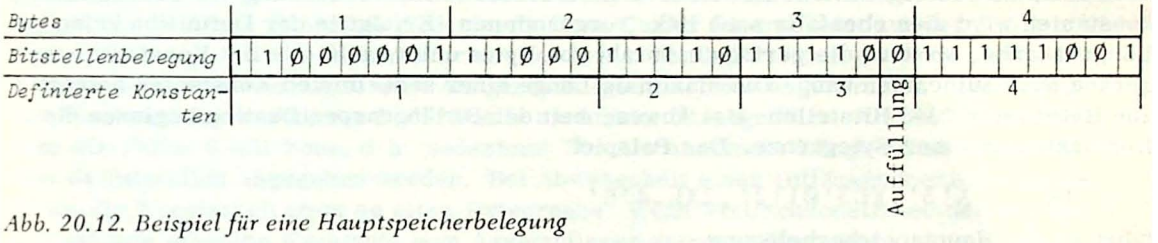


Abb. 20.12. Beispiel für eine Hauptspeicherbelegung

Die letzte der durch dieses Statement definierten vier Konstanten wird vom Assemblierer auf Bytegrenze gelegt, weil bei ihr die auf Bitstellen bezogene Längenspezifikation fehlt. Infolgedessen mußte ein „Füllbit" eingeschoben werden.

Die Wiederholungs- und Längenfaktoren können, wie bisher gezeigt, als dezimale Direktwerte angegeben werden. Sie dürfen aber auch durch in Klammern eingeschlossene Ausdrücke präsentiert werden. Alle in den Ausdrücken auftretende Symbole müssen zuvor definiert sein. Verfolgen wir auch hierzu ein Beispiel:

```
XUW     DC      CL2'MA'
XPK     DC      CL3'R'
XNL     DC      (XPK-XUW)CL.(2*(XPK-XUW+2))'D'
```

Das letzte Statement wird vom Assemblierer so aufgelöst, als ob wir

```
XNL     DC      2CL.8'D'
```

niedergeschrieben hätten.

20.10. Sedezimale und duale Konstanten

Konstanten vom X-Typ (sedezimalen Typ) besitzen den gleichen Aufbau wie die Konstanten vom C-Typ; selbstverständlich dürfen bei ihnen zwischen die Anführungszeichen nur Sedezimalziffern geschrieben werden. Beispiele:

Durch

```
DC    XL3'Ø112'
```

wird die drei Bytes umfassende Konstante ØØØ112 definiert, während durch

```
DC    2XL2'C4'
```

drei Konstanten von jeweils zwei Bytes Länge erzeugt werden, so daß also folgende Hauptspeicherbelegung entsteht: ØØC4ØØC4ØØC4.

Sedezimale Konstanten erscheinen also in den DC-Statements in der Form

`X's···s'`

oder allgemeiner in der Form

`dXLn's···s'`

Hierbei bedeuten die s sedezimale Ziffern. Ist ein Auffüllen erforderlich, so erfolgt dies nach links mit dualen Nullen. Bei einer evtl. erforderlichen Verkürzung der Sedezimalkonstanten wird dies ebenfalls nach links vorgenommen. Erfolgt in der Definition keine Längenangabe, so wird die geringste Anzahl von Bytes unterstellt, die die Konstante gerade noch aufnehmen kann. Die maximale Länge einer sedezimalen Konstanten beträgt 256 Bytes oder 2048 Bitstellen. Bei Abwesenheit der Bitlängenspezifikation beginnen die Konstanten an einer Bytegrenze. Das Beispiel

```
DC    X'3',XL3'Ø112',2XL.4'E'
```

führt zu der Hauptspeicherbelegung

```
Ø3 ØØØ112 E E
```

Insgesamt werden durch die vier Konstanten dieser Vereinbarung fünf Bytes belegt.

Duale Konstanten sind entweder in der einfachen Form

`B'z···z'`

oder in der allgemeinen Form

`dBLn'z···z'`

in die DC-Vereinbarung hineinzustellen. Konstanten vom B-Typ (duale Konstanten) ähneln den Konstanten vom X-Typ (sedezimale Konstanten) in vielen Belangen, nur müssen selbstverständlich Dualziffern z an Stelle der Sedezimalziffern geschrieben werden. Somit ist die Definition

```
DC    2BL2'11ØØØ1ØØ'
```

äquivalent der Definition

```
DC    2XL2'C4'
```

Vielfachkonstanten können auf einmal durch das Statement

`dXLn'konstante`$_1$`,···,konstante`$_m$`'`

definiert werden. Diese Definition ist gleichwertig der Definition

`dXLn'konstante`$_1$`',···,dXLn'konstante`$_m$`'`

Eine solche Definition ist sowohl für die Konstanten vom X-Typ als auch für die Konstanten vom B-Typ erlaubt, nicht jedoch für die Konstanten vom C-Typ (Zeichenkonstanten).

Im Betriebssystem DOS ist die Definition von Vielfachkonstanten vom B-Typ oder vom X-Typ auf die soeben geschilderte Weise nicht möglich. Diese Einschränkung gilt, wie wir in den folgenden Abschnitten sehen werden, jedoch nicht für alle Konstantentypen.

20.11. Weitere numerische Konstanten

Konstanten vom Z-Typ werden zur Definition von Dezimalzahlen benutzt, deren Ziffern ein Byte belegen, also neben dem Ziffernteil auch einen Zonenteil besitzen. Sie werden wie folgt festgelegt:

```
DC     dZLn'dezimalzahl1,···,dezimalzahlm'
```

Der ersten Ziffer jeder so definierten Dezimalzahl kann zusätzlich ein Vorzeichen (+ oder -) vorangehen. Wird kein Vorzeichen codiert, so wird ein Pluszeichen angenommen. Innerhalb der Dezimalzahlen kann an einer beliebigen Stelle ein Dezimalpunkt niedergeschrieben werden, doch wird er vom Assemblierer ignoriert. Bei DC-Statements der soeben geschilderten Art werden Dezimalzahlen mit Zonenteilen erzeugt: Der durch das Vorzeichen bedingte Zonenteil (Sedezimalziffern C oder D) erscheint auf der Stelle niedrigster Ordnung, alle anderen Ziffernstellen besitzen als Zonenteil die Sedezimalziffer F. Jede definierte Konstante wird als ganzzahlige Konstante angenommen. Auffüllung und Abschneidung werden im Bedarfsfalle nach links durchgeführt. Als Auffüllzeichen dient die Ziffer 0 mit Zone, d.h. sedezimal F0. Als maximale Länge können 8 Bytes oder 64 Bitstellen angegeben werden. Bei Abwesenheit einer Bitlängenspezifikation beginnen die Konstanten stets an einer Bytegrenze. Wenn Vielfachkonstanten definiert sind, so wird jede einzelne Konstante vom Assemblierer getrennt übersetzt.

Durch das Statement

```
DC     2ZL3'45,-3,+7896'
```

werden also z.B. drei Konstanten erzeugt, welche sedezimal wie folgt geschrieben werden können:

```
FØF4C5   FØFØD3   F8F9C6
```

Diese drei Konstanten werden anschließend dupliziert, so daß damit sechs Konstanten entstehen, die insgesamt 18 Bytes wie folgt belegen:

```
FØF4C5FØFØD3F8F9C6FØF4C5...
```

Das Statement

```
DC     Z'45,-3,+7896'
```

erzeugt ebenfalls drei Konstanten, doch besitzen diese eine unterschiedliche Länge:

```
F4C5 D3 F7F8F96
```

Insgesamt werden also bei diesem Beispiel 7 Bytes belegt.

Konstanten vom P-Typ sind identisch den Konstanten vom Z-Typ, nur daß die Dezimalzahlen als gepackte Dezimalzahlen gespeichert werden. Bei Abwesenheit einer Längenspezifikation wird der Konstanten die minimale Länge gegeben, die zur Aufnahme der Konstanten erforderlich ist. Somit belegt P'34' zwei Bytes; in Sedezimalziffern ausgedrückt ergibt sich also 034C.

Duale Konstanten in Festpunktdarstellung werden wie folgt definiert:

```
dFLnSsEw'konstante1,···,konstantem'
```

Der Skalenfaktor s und der Exponent w können als dezimale Direktwerte oder als Ausdrücke codiert werden. Werden Vielfachkonstanten festgelegt, so werden diese getrennt übersetzt. Die Maximallänge ist auf 8 Bytes oder 64 Bitstellen beschränkt. Bei Fehlen einer Längenspezifikation beginnt jede Konstante auf einer Wortgrenze. Erfolgt eine Längenangabe, so beginnen die Konstanten an einer Bytegrenze, bei einer Längenangabe in Bitstellen bei der nächsten verfügbaren Bitstelle. Die Konstante selbst wird wie eine dezimale Konstante niedergeschrieben und kann sowohl ein Vorzeichen als auch einen Dezimalpunkt aufweisen. Jede so fixierte Konstante wird in eine Dualzahl mit s Stellen nach dem Dualpunkt konvertiert. Vor der Konvertierung erfolgt jedoch zunächst erst die Multiplikation mit 10^w. Das Ergebnis wird gerundet und zur gespeicherten Konstanten umgeformt. Wenn s negativ angegeben wird, so schließt eine solche Festlegung ein, daß der Dualpunkt rechts von der letzten Ziffernstelle der Zahl liegt. Normalerweise werden der Skalenfaktor s und der Exponent w nicht benutzt. Werden sie hingegen aufgeführt, so muß s im Wertebereich von -187 bis +346 liegen, w im Wertebereich zwischen -85 und +75.

Erste Beispiele für Definitionen von Konstanten des F-Typs sind:

1) `DC F'19'`

Die Übersetzung führt hier zu der Speicherbelegung, sedezimal ausgedrückt,

`ØØØØ ØØ13`

2) `DC FL2'34',FL2'-7'`

Die Übersetzung führt hier zu folgender Speicherbelegung (in Sedezimalziffern):

`ØØ22 FFF9`

Dieses Statement ist äquivalent dem Statement

`DC FL2'34,-7'`

Weitere Beispiele sind in der Abb. 20.13. dargestellt.

Definierte Konstante (Operand)	*Konstante nach der Übersetzung (in sedezimaler Form)*
FS2E1'1.25'	ØØØØ ØØ32
FS2'12.5'	ØØØØ ØØ32
F'12.5'	ØØØØ ØØØD (gerundeter Bruch)
FS-1'12.5'	ØØØØ ØØØ6
FE-1'12.5'	ØØØØ ØØØ1

Abb. 20.13. Weitere Beispiele für Konstanten des F-Types

Anstelle von Konstanten des F-Typs können auch Konstanten vom H-Typ benutzt werden. Beide sind mit Ausnahme der Standardausrichtung völlig gleichwertig. Bei Fehlen der Längenangabe werden nämlich Konstanten vom H-Typ auf Halbwortgrenzen ausgerichtet. Eine typische Konstante sieht wie folgt aus:

`H'34'`

Nach der Übersetzung sieht die Hauptspeicherbelegung, sedezimal dargestellt, wie folgt aus:

`ØØ22`

Gleitpunktkonstanten sind wie folgt zu definieren:

`dELnSsEw'konstante`$_1$`,···,konstante`$_m$`'`

Die Konstanten vom E-Typ gehören zu den Gleitpunktkonstanten in Kurzdarstellung (4 Bytes) und sind auf Wortgrenzen ausgerichtet, wenn keine Länge definiert ist. Die Konstanten vom D-Typ (gleiche Form wie oben, nur beginnend mit dD···) gehören zu den Gleitpunktkonstanten in Langdarstellung (8 Bytes) und sind auf Doppelwortgrenzen ausgerichtet, wenn keine Länge definiert ist. Erfolgt dagegen eine Längenangabe, so darf diese bei beiden Konstantentypen 8 nicht übersteigen. Gleitpunktkonstanten mit erweiterter Genauigkeit werden bei der Definition durch den Buchstaben L kenntlich gemacht; ihre Definition fängt also mit dL ··· an. Bei fehlender Längenangabe werden sie auf Doppelwortgrenzen ausgerichtet. Ihre maximale Länge beträgt 16 Bytes oder 128 Bitstellen. Der Wertebereich für den in E-, D- und L-Definitionen angebbaren Konstanten ist der gleiche wie bei den Konstanten vom F- bzw. H-Typ.

Der Skalenfaktor s von Gleitpunktkonstanten steht in keiner Beziehung zum Skalenfaktor s von Festpunktkonstanten. Bei Gleitpunktkonstanten gilt folgendes: Nach der Konvertierung der definierten Konstanten in eine Dualzahl in Gleitpunktdarstellung bewirkt der Skalenfaktor s, daß bei der Mantisse eine Stellenverschiebung um so viele Sedezimalstellen nach rechts erfolgt, wie durch den Skalenfaktor angegeben sind. Die Charakteristik wird dabei gleichzeitig korrigiert, und zwar entsprechend der Zahl der verschobenen Stellen. Der Skalenfaktor kann sich deshalb nur im Wertebereich zwischen 0 und 14 (bzw. 28 bei Konstanten des L-Typs) bewegen. Durch die Verwendung eines von Null verschiedenen Skalenfaktors kann man also erreichen, daß der Assemblierer eine Gleitpunktzahl in unnormalisierter Form erzeugt. Wird der Skalenfaktor bei der Definition weggelassen, so wird standardmäßig für s der Wert 0 angenommen.

Für den Assemblierer des Betriebssystems DOS ist es erforderlich, Symbole in Skalenfaktor- und Exponentenausdrücken bereits von ihrer Verwendung zu definieren.

20.12. Adreßkonstanten

a) *Konstanten vom A-Typ*
Diese besitzen die Form

$$\boxed{\text{dALn(konstante}_1\text{,···,konstante}_m\text{)}}$$

Die Adreßkonstanten vom A-Typ können in Form beliebiger absoluter, verschieblicher oder komplex verschieblicher Ausdrücke angegeben werden. Handelt es sich um einen absoluten Ausdruck, so kann die Länge maximal 4 Bytes oder 32 Bitstellen betragen. Ist der Ausdruck nicht absolut, so muß die Länge entweder 3 oder 4 Bytes betragen. Bei fehlender Längenangabe wird eine Standardlänge von 4 Bytes unterstellt, die zugleich an eine Wortgrenze angelehnt sind. Ist dagegen die Spezifikation der Länge erfolgt, so erfolgt die Ausrichtung der Konstanten auf Bytegrenzen, es sei denn, der Längenangabe liegt die Anzahl der zu belegenden Bitstellen zugrunde. Dann allerdings beginnen die weiteren Konstanten auf der nächsten verfügbaren Bitstelle. Der Wert des Ausdruckes wird ermittelt und zu einer ganzen Dualzahl konvertiert. Nach der Konvertierung wird die Konstante rechtsbündig gespeichert und im Bedarfsfall nach links mit Nullen aufgefüllt oder abgeschnitten.

Konstanten vom A-Typ werden häufig zur Speicherung von solchen Adressen benutzt, die als Basisadressen oder Verzweigungsadressen dienen. Ebenso werden aber auch zur Definition von ganzen, ein Wort belegenden Dualzahlen eingesetzt, und zwar dann, wenn deren Wert bequem als Ausdruck niedergeschrieben werden kann. Ein Beispiel möge unsere Betrachtungen abrunden:

```
          DC      A(4)
CAESE     DC      A(*,CAESE+5,*-CAESE)
```

Die erste dieser beiden Vereinbarungen möge auf dem Speicherplatz mit der sedezimalen Adresse 010 000 liegen; die zweite beginnt dann auf dem Speicherplatz 010 004. Vom Assemblierer wird aus beiden Statements die nachfolgende Speicherbelegung erzeugt:

```
0000 0004|0001 0004|0001 0009|0000 00081
↑010000
```

Erinnern wir uns bei dieser Gelegenheit an die Bedeutung des Sternzeichens bei der Adreßbildung. Es bezieht sich stets auf den Inhalt des Adreßzählers (Zuordnungszählers) in dem Augenblick, in dem die Konstante übersetzt wird. Damit besitzt *-CAESE auch den Wert 8, da seit Entdeckung des Symbols CAESE vom Assemblierer inzwischen acht Bytes belegt wurden, nämlich die beiden Adreßkonstanten * und CAESE+5.

b) *Konstanten vom Y-Typ*

Die Konstanten vom Y-Typ sind ähnlich denen des A-Typs. Unterschiede bestehen hinsichtlich der impliziten Länge; diese beträgt bei Konstanten vom Y-Typ nämlich 2 Bytes und nicht 4 Bytes wie bei denen vom A-Typ. Außerdem erfolgt bei fehlender Längenangabe ihre Ausrichtung auf Halbwortgrenzen. Die Länge einer Konstanten vom Y-Typ darf bei Vorliegen eines absoluten Ausdruckes 2 Bytes nicht überschreiten. Bei verschieblichen oder komplex verschieblichen Ausdrücken ist eine Längenangabe von 2 Bytes die einzig erlaubte.

Adreßkonstanten vom Y-Typ werden in der Praxis nur sehr wenig benutzt.

c) *Konstanten vom V-Typ*

Die Adreßkonstanten vom V-Typ sind ähnlich den Adreßkonstanten vom A-Typ. Die Konstante muß jedoch aus einem einzigen verschieblichen Symbol bestehen (ein Ausdruck ist nicht erlaubt). Ihre Länge muß drei oder vier Bytes betragen. Bei fehlender Längenangabe wird standardmäßig eine Länge von vier Bytes angenommen, wobei diese an einer Wortgrenze ausgerichtet sind. Das zur Spezifizierung dieser Konstante benutzte Symbol muß außerhalb der für die Übersetzung vorgesehenen Programmeinheit definiert sein. Normalerweise werden Adreßkonstanten vom V-Typ dazu benutzt, um in eine Programmeinheit eine Adresse hineinzubringen, die in irgendeinem Unterprogramm oder im Hauptprogramm definiert ist.

d) *Konstanten vom S-Typ*

Sie besitzen die Form

```
S(konstante)
```

Ihre Länge beträgt stets zwei Bytes. Ist keine Länge angegeben, so wird die Konstante an eine Halbwortgrenze gebracht. Durch Adreßkonstanten des S-Typs werden Adressen in der Form Basisadresse - Distanzadresse festgelegt. Die Konstante kann in einer der nachstehenden Formen definiert werden:

```
distanz (basis)
distanz
ausdruck
```

Hierbei bedeutet distanz einen absoluten Ausdruck, der als Distanzadresse benutzt werden soll. Mit basis ist ein absoluter Ausdruck gemeint, der als Basisadresse dienen soll. Schließlich stellt ausdruck einen verschieblichen Ausdruck dar, der vom Assemblierer in das Paar Basisadresse - Distanzadresse umgeformt werden soll. Liegt für basis der Wert 0 vor, so kann distanz auch für sich allein angegeben werden.

Konstanten vom S-Typ werden dann benutzt, wenn es darum geht, den Adreßteil einer Instruktion festzuhalten. Vorsichtig ausgedrückt kann man wohl sagen, daß das in der Regel keine gute Programmiertechnik ist.

Ein Beispiel soll die Betrachtungen über die Konstanten des S-Typs abrunden. Die Codierung S(12(1) bedeutet, daß das Basisregister 1 und die Distanzadresse 12 gemeint ist. Die Speicherung erfolgt in zwei Bytes zu 100C (sedezimal).

Zum Schluß wäre zu bemerken, daß die Adreßkonstanten, auch einfach Adkons genannt, in Klammern einzuschließen sind; alle anderen Konstanten sind bekanntlich von Hochkommas umgeben.

Die Verwendung von Konstanten des Y-Typs ist im allgemeinen auf Computer beschränkt, deren Hauptspeicherkapazität nicht größer als 32 k ist, da keine höheren Adressen in einem Halbwort untergebracht werden können. Die Beispiele, die wir in früheren Abschnitten dieses Kapitels durchgesprochen haben, zeigen uns die hauptsächlichsten Anwendungsgebiete für Adreßkonstanten des S-Typs auf. Adreßkonstanten vom V-Typ werden beim Schreiben von Programmen benutzt, wenn in diesem zu einem Unterprogramm oder zu einem Programmabschnitt verzweigt werden soll, der in einem anderen Modul enthalten ist. Durch V-Typ-Konstanten werden die Adressen der Eingangsstellen in diese Unterprogramme bzw. Programmabschnitte festgehalten. Durch Laden dieser Adressen in Register können dann die entsprechenden Eingangsstellen angesprungen werden.

20.13. Literale

Literalkonstanten ermöglichen dem Programmierer die Festlegung von Konstanten direkt in einer Instruktion. Ein Literal besteht aus einer Konstanten, der ein Gleichheitszeichen vorausgehen muß. Der Assemblierer erzeugt dann selbsttätig die Konstante und stellt ihre Adresse in die Instruktion hinein. Betrachten wir dazu ein Beispiel:

Sedez. Adresse	*Instruktion*	
ØØ8 ØØØ	L	1Ø,=F'14'
ØØ8 ØØ4	B	XYZ
ØØ8 ØØ8	EOJ	
ØØ8 Ø18	...	

Unmittelbar im Anschluß an die vom generierten Code belegten Hauptspeicherplätze eines Programmes plaziert der Assemblierer die Konstanten, die in den Literalen aufgeführt sind. In diesem Fall wird die Konstante F'14' auf den Speicherplatz mit der Adresse 008018 gespeichert. Diese Adresse wird als Operand in die Ladeinstruktion eingefügt.

Werden mehrere Literale benutzt, so bedient sich der Assemblierer der folgenden Regeln hinsichtlich der Reihenfolge der Konstanten.

1) Die zum ersten Literal gehörende Konstante wird auf eine Doppelwortgrenze ausgerichtet.

2) Die Konstanten, die zu Literalen gehören, deren Länge ein Vielfaches von 8 (also 8, 16, 32 usw.) beträgt, werden zuerst gespeichert.

3) Es folgen die Konstanten, die zu Literalen gehören, deren Länge ein Vielfaches von 4 beträgt.

4) Es schließen sich die Konstanten an, die zu Literalen gehören, deren Länge ein Vielfaches von 2 beträgt.

5) Zum Schluß folgen alle übrigen Literale.

Alle in Literalen vorkommende Konstanten unterliegen den folgenden Beschränkungen:

——— Vielfache Operanden sind nicht erlaubt
——— Der Wiederholungsfaktor 0 ist nicht erlaubt
——— Wiederholungsfaktoren, Längenangaben, Skalenfaktoren und Exponenten dürfen nur als einfache Glieder auftreten

Beim Assemblierer des Betriebssystems DOS ist es nicht erlaubt, daß in Literale Konstanten vom S-Typ hineingestellt werden.

20.14. Definitionen von Datengebieten

Neben den DC-Statements, durch die konstante Daten einem Programm einverleibt werden, können wir in einem Programm auch DS-Statements verwenden. Diese verwenden wir sinnvollerweise dann, wenn es uns darum geht, Symbole zu definieren; in den reservierten Platz werden vom Assemblierer keine Daten hineingestellt. Damit können wir Hauptspeichergebiete für Eingabe/Ausgabedaten bzw. Arbeitsgebiete freihalten, ohne daß wir uns mit geeigneten Anfangswerten für diese Gebiete herumplagen müssen.

Das Statement „Definieren Speicher" (Define Storage) besitzt folgende allgemeine Form:

```
name      DS    operand1,···,operandm
```

Dieses Statement ähnelt also dem DC-Statement; im Gegensatz zu diesem werden aber in die reservierten Speicherplätze keine Daten geladen, wenn das Objektprogramm zur Ausführung in den Hauptspeicher eingespeichert wird. Jeder Operand kann genau so aufgebaut bzw. zusammengestellt werden wie bei der DC-Vereinbarung. Wenn die Längenangabe niedergeschrieben wird, braucht jedoch keine Konstante notiert zu werden. Wird auf die Angabe der Längenspezifikation verzichtet, dann ermittelt der Assemblierer aus der Konstanten, wie viele Bytes er reservieren muß. Wenn weder eine Längenangabe noch eine Konstante niedergeschrieben werden, so wird eine Länge von 1 angenommen, wenn es sich um Daten des C-, X-, B-, Z- oder P-Typs handelt; es wird dagegen eine Länge von 2 Bytes angenommen, wenn es sich um Daten des H-Typs handelt, eine Länge von 4 für Daten des F- und des E-Typs sowie eine Länge von 8 für Daten des D-Typs. Adreßkonstanten dürfen bei DS-Statements grundsätzlich nicht angegeben werden. - Definitionen von Speichergebieten der Typen C und X können mit einer Länge von bis zu 65536 Bytes getätigt werden, beim DC-Statement waren es ja nicht mehr als 256 Bytes.

Wir wollen eine Reihe von DS-Vereinbarungen betrachten:

```
ABC       DS    F
ACG       DS    7H
ARD       DS    CL24Ø
BRT       DS    C'79&&8Ø'
```

Für das Symbol ABC werden vier Bytes, beginnend an einer Wortgrenze, reserviert, für ACG werden 14 Bytes mit Halbwortbegrenzung freigehalten. Man beachte bei ACG, daß durch H eine implizite Länge von 2 Bytes festgehalten ist. Für ARD werden 240 Bytes ohne Begrenzungszuordnung freigehalten und für das letzte Symbol 5 Bytes, ebenfalls ohne Begrenzungszuordnung.

Obgleich Speicherplatz freigehalten wird, wird bekanntlich nichts in diese Speicherplätze geladen, wenn das Objektprogramm zur Ausführung in den Hauptspeicher geholt wird. Der Programmierer darf sich deshalb von keinen Voraussetzungen leiten lassen, daß diese Bereiche irgendwelche bestimmten Werte enthalten. Insbesondere darf keinesfalls

angenommen werden, daß diese Bereiche Nullen enthalten. Um gewünschte bzw. notwendige Begrenzungszuordnungen zu erhalten, werden vom Assemblierer zwar Bytes übersprungen, doch in keiner Weise mit irgendwelchen bestimmten Werten gefüllt.

Durch DS-Statements kann man eine Ausrichtung der Gebiete auf bestimmte Grenzen erzwingen. Beispiel:

```
BRT      DS    C'79&&8Ø'
KL       DS    ØD
KPR      DS    C'STS.'
```

Da eine D-Typangabe eine Ausrichtung auf eine Doppelwortgrenze erfordert, beginnen die für KPR freigehaltenen Bytes mit einem Byte, dessen Adresse durch 8 teilbar ist. Für KL werden keine Bytes reserviert; nur der Zuordnungszähler wird gezwungenermaßen auf einen durch acht teilbaren Wert gebracht.

Ein anderer Zweck des DS-Statements mit einem Wiederholungsfaktor 0 besteht darin, einem Symbol eine bestimmte implizite Länge mitzugeben. Beispiel:

```
KARTE    DS    ØCL8Ø
FELD1    DS    CL5
FELD2    DS    CL27
FELD3    DS    CL48
```

Bei einer solchen Definition können wir das Symbol KARTE beispielsweise in einer Instruktion verwenden, in der es darum geht, 80 Bytes zu übertragen. Dabei ersparen wir uns die Eintragung der Längenangabe in die Instruktion.

Der Leser mag sich vielleicht schon die Frage gestellt haben, warum in DS-Statements Konstanten aufgeführt werden können. Der Grund ist einfach: Der Programmierer braucht nur die Konstante niederzuschreiben, wodurch er sich das mühselige Abzählen der Bytes ersparen kann, wenn er eine entsprechende Anzahl von Bytes im Hauptspeicher vom Assemblierer reservieren lassen will. Meist enthält allerdings eine DS-Vereinbarung nur den Wiederholungsfaktor und die Typangabe; bei den Typen C, X oder B wird dann noch eine Längenangabe folgen.

20.15. Das CCW-Statement

Kanalbefehlswörter [1] können sehr leicht einem Programm eingegliedert werden, wenn man sich des folgenden Statements bedient:

```
name     CCW   befehlsschlüssel,adresse,anzeiger,zähler
```

Durch dieses Statement wird ein Kanalbefehlswort aufgesetzt, das an einer Doppelwortgrenze beginnt. Unter befehlsschlüssel muß ein absoluter Ausdruck erscheinen; dieser bestimmt den Inhalt des Bytes, das den Befehlsschlüssel aufnimmt. Unter adresse kann ein absoluter oder ein verschieblicher Ausdruck aufgeführt werden. Durch ihn wird die Datenadresse spezifiziert, die in die Bytes 1, 2 und 3 des CCW gespeichert werden soll. Unter anzeiger versteht man einen absoluten Ausdruck. Durch ihn werden die Anzeigerstellungen im Byte 4 des CCW vorgegeben. Die letzten drei Bitstellen dieses Bytes müssen den Inhalt 0 aufweisen. Durch zähler werden die letzten beiden Bytes des CCW angesprochen. Der unter ihm zu spezifizierende absolute Ausdruck gibt die Zahl der in die EA verwickelten Bytes an. Das Byte 5 des CCW wird durch dieses Statement immer auf 0 gesetzt.

---

1) Channel Command Words

Mit einigen typischen CCW-Statements wollen wir die Betrachtungen abschließen:

```
NAV     CCW    X'Ø2',RODINAL,Ø,8Ø
        CCW    2,*+8,X'2Ø',8Ø
```

Jedes der beiden hier dargestellten Kanalbefehlswörter veranlaßt das Lesen einer 80spaltigen Lochkarte.

Aufgabengruppe 55

1) Schreibe in der Maschinensprache das Programm nieder, das vom Assemblierer für die Aufgabe 1) der Aufgabengruppe 54 generiert wird. Ersetze zuvor jedes Makro durch das Statement

```
DS      CL16
```

2) Schreibe in der AL ein Programm, das das folgende Problem löst: Dem Programm wird im Register 1 eine nichtnegative ganze Zahl N übergeben. Es soll ein 100 Bytes langes Feld aufgebaut werden, das beim Speicherplatz mit der Adresse LOC beginnt und dessen Inhalt durch den Wert von N bestimmt ist. Wenn N 100 ist, so sollen alle Bytes des Feldes nur Leerzeichen enthalten; ist N dagegen kleiner als 100, so sollen in den ersten N Stellen des Feldes Leerzeichen stehen, danach folgt ein Sternzeichen, dem sich wiederum 99-N Leerzeichen anschließen. Der Ausgang aus diesem Programm soll durch eine Verzweigung zu dem Speicherplatz erfolgen, dessen Adresse im Register 14 steht.

3) Es sind die folgenden Daten gespeichert:

003 400 bis 003 405	Bruttolohn
003 406 bis 003 40B	Lohnsteuer
003 40C bis 003 411	Sozialabgaben
003 412 bis 003 417	Krankenversicherung

Alle Felder sind für gepackte Dezimalzahlen bestimmt. Schreibe ein Programm zur Berechnung des Nettolohnes; dieser soll auf die Speicherplätze 003 418 bis 003 41D gespeichert werden.

4) Für die folgenden Daten sind sechs Bytes lange Felder vorgesehen, die gepackte Dezimalzahlen aufnehmen können. Wenn nicht anders angegeben, enthält jedes Feld zwei Dezimalstellen:

002 500	Normale Arbeitszeit
002 506	Überstunden
002 50C	Normaler Stundenverdienst (in Dollars mit vier Dezimalstellen)
002 512	Krankengeld
002 518	Bruttolohn

Berechne in einem Programm aus den übrigen Werten den Bruttolohn und speichere ihn auf dem angegebenen Speicherplatz! Für Überstunden wird das 1,5-fache des Normalverdienstes gezahlt.

5) Schreibe ein Unterprogramm zur Lösung der folgenden Aufgabenstellung:
Dem Programm wird eine Zahl in Gleitpunktdarstellung (N) als Argument übergeben; das Argument befindet sich im Gleitpunktregister 0 und besitzt einfache Genauigkeit. Ins allgemeine Register 1 soll 0 gestellt werden, wenn $|N| \geq 10^9$ ist, anderenfalls ist der ganzzahlige Anteil von N, umgewandelt in eine duale Festpunktzahl, in das gleiche Register zu stellen.

6) Ändere die Aufgabe 5) wie folgt ab: In die allgemeinen Register 0 und 1 ist der ganzzahlige Anteil von N in Form einer acht Byte langen gepackten Dezimalzahl hineinzustellen.

7) Man sagt, daß „a kongruent b modulo c" ist und wir schreiben

   a=b (mod c),

   wenn a-b durch c teilbar ist. Hierbei sind a, b und c ganze Zahlen.

   In der Zahlentheorie wird gezeigt, daß die Gleichung

   ax=b (mod c)

   eine und nur eine ganzzahlige Lösung für x im Intervall zwischen 0 und c-1 besitzt, wenn c eine Primzahl ist.

   Schreibe ein Programm in der AL, das Werte für a, b und c einliest und drucke alle Werte von x aus, die die Gleichung erfüllen und im Intervall von 0 bis einschließlich c-1 liegen. Beginnen wir damit, daß wir x der Reihe nach auf die Werte 0, 1, 2, ···, c-1 setzen und das Produkt aus x und a bilden. Ist dieses Produkt größer als c, so subtrahieren wir solange c von diesem Produkt, bis wir einen Wert erhalten, der kleiner als c ist. Diesen Wert vergleichen wir mit b; finden wir eine Gleichheit, so haben wir eine Lösung ermittelt. Die Rechnung soll für die folgenden Beispiele durchgeführt werden:

   a) a=3, b=7, c=13; b) a=13, b=28, c=69

8) Schreibe ein Programm in der AL, das eine Lochkarte mit den Werten von a, b, c und m einliest und bestimme die Lösungen für die Gleichung

   $ax^2 + bx + c = 0 \pmod{m}$

   Die Lösungen sind auszudrucken. Beschränke hierbei a, b, c und m auf dreistellige ganze Zahlen. Setze weiterhin voraus, daß sowohl a als auch b als auch c im Wertebereich zwischen 0 bis einschließlich m-1 liegen. Alle arithmetischen Operationen sind mit Dualzahlen in der Festpunktdarstellung durchzuführen.

9) Schreibe ein Programm in der AL, das folgende Divisionsaufgaben löst:
   Der Dividend steht als gepackte vierstellige Dezimalzahl auf den Speicherplätzen von 003 500 bis 003 502 und der Divisor steht als vierstellige Dezimalzahl auf den Speicherplätzen von 003 503 bis 003 505. Der Quotient ist auf das Feld zu speichern, dessen Bytes die Adressen 003 506 bis 003 508 besitzen, der Rest auf das Feld, dessen Bytes die Adressen 003 509 bis 003 50B aufweisen.

20.16. Steueranweisungen für den Assemblierer

In diesem Abschnitt wollen wir eine Reihe von weiteren Steueranweisungen für den Assemblierer kennenlernen, die für manche Zwecke recht nützlich sein können. (Wie bereits erwähnt, bezeichnen wir solche Steueranweisungen auch als Vereinbarungen, da durch sie vereinbart wird, in welcher Weise der Assemblierer arbeiten soll.)

Durch vier Statements kann das Format der vom Assemblierer produzierten Druckausgabe gesteuert werden. Sie beeinflussen den Ablauf des Objektprogrammes nicht. Diese Vereinbarungen lauten:

1) *Überschrift (Title)*
Die allgemeine Form dieses Statements lautet wie folgt:

```
name      TITLE  'text'
```

Das Namensfeld kann bis zu acht alphabetische oder numerische Zeichen [1] enthalten. Der Inhalt dieses Feldes wird nicht in die Symboltabelle aufgenommen. Die Zeichen werden vielmehr in die Spalten 73 bis 76 aller vom Assemblierer erzeugten Ausgabekarten gestanzt. Ausgenommen hiervon sind alle Karten, die durch ein REPRO- oder PUNCH-Statement erzeugt werden. Im Operandenfeld kann zwischen die Anführungszeichen eine Zeichenfolge gesetzt werden, die bis zu 100 Zeichen umfaßt. Die Zeichen dieser Folge werden solange auf die Kopfzeile jeder vom Assemblierer produzierten Seite gedruckt, bis ein neues TITLE-Statement festgestellt wird. Jedes neue TITLE-Statement bewirkt, daß die Druckausgabe auf einer neuen Seite fortgesetzt wird. Natürlich dürfen alle TITLE-Statements einen Namen im Namensfeld aufweisen, doch nur das erste nichtleere Namensfeld aller TITLE-Statements wird beachtet, die anderen fallen unter den Tisch. - Der Assemblierer erzeugt automatisch Reihenfolgenummern für alle von ihm produzierten Lochkarten; sie erscheinen auf den Karten rechtsbündig auf die Spalte 80 ausgerichtet. Sie reichen nach links bis zur ersten nichtleeren Spalte, die sich als Resultat eines TITLE-Statements mit nichtleerem Namensfeld ergibt.

2) *Beginn einer neuen Seite (Eject)*
Allgemein lautet die Form dieses Statements

```
leer      EJECT leer
```

Namensfeld und Operandenfeld bleiben bei diesem Statement leer. Das Auftreten dieses Statements bewirkt, daß die nächste Zeile der Ausgabeliste auf eine neue Seite gedruckt wird.

3) *Überspringen von Zeilen (Space)*
Die allgemeine Form dieses Statements lautet:

```
leer      SPACE leer
```

oder

```
leer      SPACE ganze Dezimalzahlen
```

Diese Steueranweisung hat die Aufgabe, Leerzeilen in die Ausgabelisten einzufügen. Bleibt das Operandenfeld leer, so wird eine Leerzeile zwischen zwei aufeinanderfolgenden Zeilen erzeugt. Steht dagegen im Operandenfeld eine ganze Dezimalzahl, so werden zwischen die einzelnen Druckzeilen so viele Leerzeilen geschoben, wie durch die ganze Dezimalzahl spezifiziert sind. Überschreitet bei der Auflistung die eingetragene Zahl die noch auf einer Seite vorhandenen Zeilen, so wird das SPACE-Statement wie ein EJECT-Statement behandelt.

4) *Wahlweises Drucken von Daten (Print Optional Data)*
Die allgemeine Form dieser Assembliereranweisung lautet:

```
leer      PRINT   1 bis 3 Operanden
```

Maximal können bis zu drei Operanden angegeben werden, die aus den folgenden auszuwählen sind:

ON	Es soll eine Liste des Programmes gedruckt werden
OFF	Es soll keine Programmliste gedruckt werden

1) Für den Assemblierer des Betriebssystemes DOS sind maximal nur vier alphabetische oder numerische Zeichen erlaubt.

GEN	Es sollen alle Statements gedruckt werden, die bei der Auflösung von Makrostatements vom Assemblierer eingefügt werden
NOGEN	Es werden nur die Makrostatements gedruckt, jedoch nicht die Instruktionen, in die sie aufgelöst werden
DATA	Alle Konstanten sollen vollständig gedruckt werden
NODATA	Es werden nur die ersten acht Sedezimalziffern der erzeugten Konstanten gedruckt

Die Wirkung einer PRINT-Vereinbarung bleibt solange bestehen, bis ein neues PRINT-Statement vom Assemblierer gefunden wird. Zu Beginn einer Programmübersetzung nimmt der Assemblierer an, daß folgende Operanden gegeben sind:

```
PRINT    ON,NODATA,GEN
```

Bei diesem Statement können die Operanden in einer beliebigen Reihenfolge niedergeschrieben werden.

Abschließend sei noch einmal betont, daß die Drucksteueranweisungen nur die Auflistung des Programmes beeinflussen, jedoch in keiner Weise das Objektprogramm verändern.

Im Betriebssystem OS können zusätzlich die folgenden Statements zur Steuerung der vom Assemblierer gesteuerten Druckausgabe verwendet werden:

5) `leer  POP  1 oder 2 Operanden`

6) `leer  PUSH  1 oder 2 Operanden`

Die beiden möglichen Operanden sind durch die Wörter USING und PRINT gekennzeichnet. Werden beide zusammen codiert, so ist es gleichgültig, in welcher Reihenfolge sie erscheinen.

Der Operand PRINT im PUSH-Statement bewirkt, daß der augenblickliche Zustand der Druckausgabe (bestimmt durch die Operanden des Statements PRINT) gespeichert wird, wodurch spätere Bezugnahmen auf ihn ermöglicht werden. PUSH mit dem Operanden USING hat zur Folge, daß der gegenwärtige Zustand der verfügbaren Basisregister gespeichert wird.

POP bewirkt, daß der gegenwärtige Zustand der Druckausgabe (Operand PRINT) bzw. der Basisregister (Operand USING) verlorengeht. An ihre Stelle treten die Zustände, die durch das letzte PUSH-Statement mit dem gleichen Operanden weggespeichert wurden. Bis zu vier Zustände jeder Art können aufbewahrt werden. Durch jedes PUSH-Statement wird auf den Zustandsstapel ein neuer Baustein gelegt, jedes POP-Statement nimmt den Zustandsbaustein von der Spitze des Stapels weg und entfernt ihn damit auch vom Stapel, so daß beim nächsten POP-Statement ein entsprechend verkleinerter Stapel vorliegt.

Die weiteren Steueranweisungen für den Assemblierer beeinflussen die Bearbeitung des Quellenprogrammes durch den Assemblierer. Im einzelnen handelt es sich um folgende Vereinbarungen:

1) *Kopieren der nächsten Karte (Reproduce)*
Das Statement

```
leer    REPRO    leer
```

bewirkt, daß die nächstfolgende Eingabekarte kopiert und somit in das Objektprogramm aufgenommen wird. REPRO-Assembliereranweisungen können an beliebiger Stelle im Quellenprogramm erscheinen; gewöhnlich liegen sie jedoch am Anfang oder am Ende eines Programmes. Beispielsweise kann damit eine Steuerkarte, die für den Binder bestimmt ist, weitergegeben werden.

2) *Stanzen einer Karte (Punch)*
Ihre allgemeine Form sieht wie folgt aus:

```
leer      PUNCH   'zeichen'
```

Die zwischen den Anführungszeichen im Operanden stehenden Zeichen (maximal 80) werden in eine Karte gestanzt. Eine Verarbeitung der Zeichen erfolgt nicht.

Dieses Statement kann nur für die Assemblierer aufgesetzt werden, die ins Betriebssystem OS integriert ist.

3) *Eingabeformat-Steuerung (Input Format Control)*
Die allgemeine Form dieses Statements sieht wie folgt aus:

```
leer      ICTL    b,e,c
leer      ICTL    b,e
leer      ICTL    b
```

Durch b wird die Spalte festgelegt, auf der die Statements des Quellenprogrammes beginnen sollen, im allgemeinen ist das die Spalte 1. Durch e wird die Spalte festgelegt, bis zu der die Statements des Quellenprogrammes geschrieben und abgelocht werden können. Durch c kann schließlich die Folgespalte festgelegt werden, d.h. die Spalte, ab der die Statements auf weiteren Zeilen bzw. Karten fortgesetzt werden können. Meist wird hierzu die Spalte 16 herangezogen. Wird c weggelassen, so nimmt der Assemblierer an, daß keine Folgekarten eingegeben werden. Fehlt e, so nimmt der Assemblierer an, daß die Spalte 71 die Statements beendet. Ist überhaupt keine ICTL-Assembliereranweisung vorhanden, so unterstellt der Assemblierer eine solche mit dem Inhalt

```
ICTL  1,71,16
```

Die für b, e und c angegebenen Werte müssen innerhalb der folgenden Wertebereiche liegen:

$$1 \leqq b \leqq 40$$
$$2 \leqq c \leqq 40$$
$$41 \leqq e \leqq 80$$

Wird dieses Statement benutzt, so muß es als erstes Statement des Quellenprogrammes erscheinen. - In diesem Zusammenhang sei ein Verweis auf die Abb. 20.9. gestattet.

4) *Eingabe-Folgeprüfung (Input Squence Checking)*
Die allgemeine Form dieses Statements ist

```
leer      ISEQ    leer
```

oder

```
leer      ISEQ    1,r
```

Durch l bzw. r (beide müssen ganze Dezimalzahlen sein) werden die linke und die rechte Kartenspalte angegeben, die das Feld begrenzen, dessen Inhalt geprüft werden soll, um festzustellen, ob sich die Eingabekarten in aufsteigender Reihenfolge befinden. Die Folgeprüfung beginnt mit der Karte, die auf die ISEQ-Karte folgt. Die Folgeprüfung wird beendet, wenn eine ISEQ-Karte gefunden wird, deren Operandenfeld leer ist. Die Spalten, die zur Prüfung herangezogen werden sollen, dürfen sich nicht innerhalb des Statementfeldes befinden, d.h. normalerweise nicht innerhalb des Feldes, das von den Kartenspalten 1 und 72 begrenzt wird. Gewöhnlich werden zur Folgeprüfung nur die Spalten 77 bis 80 verwendet.

5) *Ende (End)*
Das letzte Statement eines in der AL geschriebenen Programmes muß das folgende Statement sein:

```
leer      END   leer
leer      END   verschieblicher Ausdruck
```

Wenn ein Operand niedergeschrieben wird, so bewirkt dieser, daß für das Objektprogramm eine Instruktion generiert wird, die eine Verzweigung zu dem Speicherplatz hervorruft, dessen Adresse durch den Ausdruck spezifiziert ist.

Diese Verzweigung hat zur Folge, daß nach dem Laden des Objektprogrammes die erste Instruktion des auszuführenden Programmes von dem angesprungenen Speicherplatz abgeholt wird. Um diese Funktion erfüllen zu können, muß die Adresse des ermittelten Speicherplatzes an den Binder (Linkage Editor) weitergegeben werden. Besteht nun ein Programm aus mehreren Programmeinheiten, die alle getrennt übersetzt (assembliert oder kompiliert) wurden, so benutzt der Binder diejenige Ansprungadresse, die er zuerst findet. Findet er keine solche, so kommt es bei der Programmausführung dazu, daß zu Beginn die Programmablaufsteuerung an das erste Byte des ersten vom Binder bearbeiteten Moduls übertragen wird.

Statements eines Quellenprogrammes, die nach dem END-Statement erscheinen, werden nicht mehr vom Assemblierer bearbeitet.

Für das Betriebssystem OS gilt zusätzlich noch folgende Regel: Fehlt das END-Statement bei einem Programm, so nimmt der Assemblierer selbständig ein solches an, allerdings mit leerem Operandenfeld.

6) *Kopieren (Copy)*
Diese Steueranweisung besitzt folgende allgemeine Form:

```
leer      COPY   symbol
```

Das Symbol muß der Name einer Datenmenge aus der Systembibliothek sein. Die durch dieses Symbol angesprochenen Daten werden aus der Bibliothek geholt und in das Quellenprogramm an der Stelle eingefügt, an der sich das COPY-Statement befindet. Dieses Statement versetzt uns also in die Lage, bei erforderlichen Wiederholungen von Programmteilen diese nur einmal zu schreiben, sei es nun im gleichen Programm oder in einem anderen.

Hinsichtlich des zu codierenden Symbols ist bezüglich der beiden Betriebssysteme folgendes zu bemerken:

a) Betriebssystem DOS
Unter symbol ist ein Buchname von SYSSLB aufzuführen, dem vom Assemblierer die Zeichenfolge A. vorangestellt wird. Liegt keine Zuweisung für SYSSLB vor, so ist SYSRES anzusprechen.

Das heißt also, daß z.B. das Statement

```
COPY   XYZ
```

bewirkt, daß das Buch

```
A.XYZ
```

aus SYSSLB oder SYSRES in das Quellenprogramm an der Stelle eingefügt wird, an der dieses Statement erscheint. Zur Aufnahme der Bücher in die Quellenstatement-Bibliothek gibt es Dienstprogramme, die von der IBM offeriert und gewartet werden.

b) Betriebssystem OS

Unter symbol ist hier der Name des Gliedes einer untergliederten Datenmenge aufzuführen, das durch das DD-Statement für SYSLIB angesprochen wird.

Die katalogisierten IBM-Prozeduren sehen SYS1.MACLIB als Datenmengenname von SYSLIB vor.

7) *Gleichsetzen (Equate)*

Die allgemeine Form dieser Steueranweisung lautet:

```
name      EQU   ausdruck
name      EQU   ausdruck,länge,typ
```

Mit diesem Statement können zusätzlich Symbole in die Symboltabelle aufgenommen werden.

Unter ausdruck kann ein absoluter oder ein verschieblicher Ausdruck codiert werden. Der Wert dieses Ausdrucks wird dem Symbol name zugewiesen; dieses Symbol erwirbt gleichzeitig auch die Länge und den Typ des ersten Symbols, das im Ausdruck aufgeführt ist. Ist dieses ein Sternzeichen oder der Name einer Instruktion, so ist die implizite Länge gleich 1. Wird der zweite Operand niedergeschrieben, so muß er als absoluter Ausdruck mit einem Wert zwischen 0 und 65535 erscheinen. Dieser Wert gilt dann als Längenattribut für das neue Symbol name. Durch den dritten Operanden, der einen absoluten Ausdruck mit einem Wert zwischen 0 und 255 darstellen muß, kann der Typ festgelegt werden, der dem neuen Symbol zugeordnet sein muß. Praktisch benutzt man zur Typfestlegung den zum Typ gehörenden Kennbuchstaben im EBCDI-Code. Das nachfolgende Beispiel stellt eine komplette Einführung eines neuen Symbols vor:

```
MARIANNE EQU   KAROLINE,8Ø,C'A'
```

Dieses Statement bewirkt, daß MARIANNE neu in die Symboltabelle aufgenommen wird, und zwar als Konstante des A-Typs (Adreßkonstante) mit der impliziten Länge 80.

Im Betriebssystem DOS ist die Verwendung der beiden letzten Operanden länge und typ nicht möglich.

Zwei typische Beispiele für die Vereinbarung EQU sollen die bisherigen Erörterungen abschließen:

```
R3        EQU   3
AKON      EQU   ABC+DEF-GHI
          USING *,R3
          MVI   AKON,X'56'
```

Geben wir noch einige Kommentare zu dieser Statementfolge. Durch Einführung von Symbolen für Register (hier: R_3 für Register 3) enthält die vom Assemblierer ausgegebene Symboltabelle auch die in Instruktionen getätigten Referenzen auf die betreffenden Register, sicher eine nützliche Hilfe beim Testen und beim Fehlersuchen in Programmen.

Im Betriebssystem OS ist die Verwendung einer weiteren Vereinbarung möglich, die in folgenden Formen auftreten darf:

```
opa       OPSYN   opb
opa       OPSYN   leer
```

Die erste Form bewirkt, daß der im Namensfeld niedergeschriebene Operationsschlüssel opa durch den im Operandenfeld stehenden opb ersetzt wird. Somit ermöglicht es uns die Vereinbarung

```
XYZ     OPSYN SRP
```

den Operationsschlüssel XYZ an Stelle des Operationsschlüssels SPR zu benutzen. Die Instruktion

```
        SRP   ABC,DEF,7
```

gilt damit für den Assemblierer als gleichwertig der Instruktion

```
        XYZ   ABC,DEF,7
```

Benutzen wir die zweite Form, d.h. lassen wir das Operandenfeld leer, so verursacht die Vereinbarung OPSYN, daß der im Namensfeld angegebene Operationsschlüssel seine Gültigkeit verliert. Somit verhüten wir beispielsweise durch das Statement

```
SRP     OPSYN
```

die Verwendung der Instruktion SRP in einem Programm.

Dem Statement OPSYN dürfen in einem Quellenprogramm nur die Statements ISEQ, ICTL, TITLE, PRINT, SPACE und EJECT vorausgehen.

Einige weitere Vereinbarungen besprechen wir im nächsten Abschnitt im Zusammenhang mit dem Zuordnungszähler.

20.17. Der Zuordnungszähler

Die Statements des Quellenprogrammes müssen dem Assemblierer zur Übersetzung zweimal zugeführt werden. Während des ersten Durchlaufs bearbeitet der Assemblierer zwar die Instruktionen, die Konstanten usw., aber übersetzt die Operanden noch nicht in die Maschinensprache, da er ihre Adressen noch nicht kennt. Stattdessen wird vorerst ein Zuordnungszähler, auch Adreßzähler genannt, eingerichtet, dem gleichzeitig ein Anfangswert zugewiesen wird. Dieser ist entweder gleich Null oder gleich dem Wert, der im START-Statement erscheint. Jedes im Namensfeld eines Statements auftretende Symbol wird in eine Symboltabelle hineingestellt; ergänzt wird die Aufnahme eines Symboles durch die gleichzeitige Aufnahme seines Wertes, seiner Länge usw. Nachdem ein Statement auf diese Weise verarbeitet ist, wird der Zuordnungszähler um den Wert erhöht, der gleich der Anzahl der Bytes ist, die die betreffende Maschineninstruktion belegen wird, bzw. die für eine Konstante oder ein Speichergebiet reserviert werden müssen. Müssen Bytes wegen einer erforderlichen Begrenzungszuordnung ausgelassen werden, so wird der Zuordnungszähler ebenfalls entsprechend der Anzahl der übersprungenen Bytes weitergesetzt. Somit enthält der Zuordnungszähler immer die Adresse des nächsten freien, d.h. nicht belegten Bytes; daher auch der Begriff „Adreßzähler".

Der Programmierer kann durch einige Assembliereranweisungen (Vereinbarungen) Einfluß auf den Wert des Zuordnungszählers nehmen.

a) *Setzen Zuordnungszähler (Set Location Counter)*
Dieses Statement besitzt allgemein folgende Form:

```
name    ORG   leer
```

oder

```
name    ORG   verschieblicher ausdruck
```

Durch dieses Statement [1] kann dem Zuordnungszähler ein neuer Wert zugewiesen werden. Wird ein verschieblicher Ausdruck aufgeführt, so darf dieser keinen Wert repräsentieren, der kleiner ist als die Adresse des Programmanfangspunktes (d.h. der Adresse, die dem entsprechenden Basisregister zugewiesen ist). Einige typische ORG-Vereinbarungen mögen noch als Beispiel folgen:

```
ORG   *-24Ø
ORG   ABC+8
ORG   GEB-26
```

Wird das Operandenfeld leer gelassen, so wird der Zuordnungszähler auf einen Wert gesetzt, der um 1 größer als die Adresse des Speicherplatzes ist, der vom Programm bis zum Auftreten dieses Statements als letzter belegt worden ist. Erscheinen also z.B. nach

```
ORG   *-24Ø
```

mehrere Instruktionen oder Konstanten bzw. Speicherreservierungen, so kann man durch

```
ORG
```

den Zuordnungszähler auf seinen alten Wert zurücksetzen.

Im allgemeinen wird das ORG-Statement dazu benutzt, um Daten anderen Daten zu überlagern. Ein typisches Beispiel zeigt uns die nachstehende Statementfolge:

```
TS    *+1
ORG   *-3
DC    X'FF'
ORG
```

In der TS-Instruktion wird bei der Übersetzung das Byte mit den Direktdaten zunächst mit dem Wert X'ØØ' gefüllt; danach wird durch die beiden Statements ORG und DC sein Inhalt mit X'FF' überlagert. Eine solche Überlagerung funktioniert naturgemäß nicht, wenn eine Adreßkonstante mit einem verschieblichen oder komplex verschieblichen Ausdruck überlagert werden soll. Der Grund hierfür ist einleuchtend: Die Verschiebung wird vom Binder (Linkage Editor) erst nach der vom Assemblierer besorgten Übersetzung vorgenommen. Die Überlagerung erfolgt vor der Verschiebung; das sollte man sich stets vor Augen halten. Doch verfolgen wir hierzu ein Beispiel:

```
DC    A(XYZ)
ORG   *-4
DC    F'35'
```

Durch diese Statementfolge wird 0000 0023 auf die Adreßkonstante von XYZ gelegt. Vor der Ausführung des Programmes, doch erst nach der Übersetzung, findet die Verschiebung statt, so daß der Wert der Adreßkonstanten gleich der Summe von 0000 0023 und dem Verschiebungsfaktor ist. Sehen wir einmal von der Überlagerung ab, so erinnern wir uns doch sicher daran, daß bei der Programmausführung die Adreßkonstante einen Wert besitzt, der gleich ist der Summe aus der Adresse von XYZ relativ zu Programmbeginn und dem Betrag der Verschiebung.

Bei der Verarbeitung der Statements verfolgt der Assemblierer auch das Auftreten von Literalen und merkt sich diese. Den bis zu einer gewissen Programmstelle aufgetretenen Literalen können Speicherplätze durch die nachfolgend beschriebene Vereinbarung zugewiesen werden, wobei diese von der nächsten verfügbaren Doppelwortbegrenzung an bereitgestellt werden.

---

1) Im Betriebssystem DOS muß das Namensfeld frei bleiben.

b) *Beginne Literalpool (Begin Literal Pool)*
Die allgemeine Form dieses Statements sieht wie folgt aus:

```
name        LTORG    leer
```

Wird unter name ein Symbol niedergeschrieben, so wird es in die Symboltabelle aufgenommen, wobei ihm die Adresse des ersten Bytes des Literalpools entspricht; außerdem wird ihm ein Längenattribut von 1 zugeordnet.

Die nicht durch ein LTORG-Statement erfaßten, im Programm verbleibenden restlichen Literale werden erst dann übersetzt und gespeichert, wenn das END-Statement festgestellt wird. Tritt ein Literal mehr als einmal vor dem LTORG auf, so wird es dennoch in den Literalpool nur ein einziges Mal hineingestellt. Zwei Literale, die an und für sich den gleichen Wert repräsentieren, jedoch durch verschiedene Konstanten dargestellt sind, werden beide im Pool festgehalten. Somit führen

```
=F'34'
```

und

```
=A(34)
```

zu zwei Eintragungen im Pool. Nach der durch das LTORG-Statement veranlaßten Herausgabe der bisher angesammelten Literale in einen Pool beginnt der Assemblierer sofort mit dem Aufbau eines neuen Pools. Wenn also =F'34' vor und nach einem LTORG-Statement erscheint, so wird dieses Literal auch in zwei Pools aufgenommen.

Im zweiten Durchlauf des Assemblierers können alle Instruktionen in die Maschinensprache übersetzt werden, da zu dieser Zeit dann jedes Symbol, das in einem Operanden auftritt, definiert in der Symboltabelle vorliegt. Fehlernachrichten werden ausgegeben, wenn in einer Instruktion ein undefiniertes Symbol gefunden wird.

c) *Bedingte Erzeugung von operationslosen Instruktionen (Conditional no operation)*
Als letzte Vereinbarung wollen wir in diesem Abschnitt das CNOP-Statement kennenlernen. Es besitzt folgende Form [1]:

```
name        CNOP     b,w
```

Durch die Operanden b und w wird ein Wertepaar spezifiziert. Sowohl b als auch w können in Form absoluter Ausdrücke angegeben werden, doch ist zu beachten, daß in diesen nur solche Symbole auftreten, die zuvor bereits definiert sind. Die Ausdrücke für b und für w müssen folgende Wertepaare repräsentieren:

```
0,4   0,8   2,4   2,8   4,8   6,8
```

Beim ersten Wertepaar wird der Zuordnungszähler auf das erste Byte eines Wortes gesetzt; ist b=2 und w=4, so wird er auf das dritte Byte eines Wortes gesetzt. Ist w=8, dann wird der Zuordnungszähler auf das erste, dritte, fünfte oder siebente Byte eines Doppelwortes gesetzt, je nachdem, ob b gleich 0, 2, 4 oder 6 ist. Befindet sich der Zuordnungszähler bereits zufällig auf einer der spezifizierten Begrenzungen, so hat die Codierung von CNOP keine Wirkung; in allen anderen Fällen werden aus der CNOP-Steueranweisung NOPR-Instruktionen erzeugt, und zwar genau so viele, daß die bis zur angegebenen Grenze fehlenden Bytes durch sie ausgefüllt werden. Je zwei übersprungene Bytes werden also mit 07 00 belegt. Wird das CNOP-Statement nach einer DC- oder einer DS-Vereinbarung benutzt, so ist es durchaus auch möglich, daß nur ein Byte übersprungen werden muß, um die verlangte Begrenzung einzuhalten. Tritt ein solcher Fall auf, so wird das Byte mit 00 belegt. Um diesen Vorgang durch einige

1) Im Betriebssystem DOS ist die Angabe eines Namens im Namensfeld nicht erlaubt.

Beispiele erläutern zu können, nehmen wir an, daß der Zuordnungszähler den Wert 8194 enthält. In diesem Fall hat

```
CNOP  2,8
```

keine Wirkung und wird einfach vernachlässigt, da der Zuordnungszähler bereits auf der gewünschten Begrenzung steht, d.h. daß er das dritte Byte eines Doppelwortes kennzeichnet.

Durch

```
CNOP  4,8
```

wird jedoch auf den Speicherplätzen 8194 und 8195 die Instruktion 07 00 gespeichert und der Zuordnungszähler auf den Wert 8196 gesetzt. Durch

```
CNOP  Ø,8
```

wird schließlich bewirkt, daß vom Assemblierer drei NOPR-Instruktionen auf die Speicherplätze von 8194 bis 8199 gespeichert werden und der Zuordnungszähler auf die nächstfolgende Doppelwortadresse, nämlich 8200 gesetzt wird.

Hauptzweck des CNOP-Statements ist es wohl, dafür Sorge zu tragen, daß Konstanten vom C- oder X-Typ durch den Assemblierer auf eine geeignete Grenze gelegt werden, wenn eine solche Konstante in einer Instruktion vom RX-Format verwendet wird. Die Möglichkeit, CNOP-Statements verwenden zu können, ist auch dann von Nutzen, wenn eine Aufrufroutine erzeugt werden soll, die aus einer Verbindung zu einem Unterprogramm und aus einer Folge von Argumenten, wie z.B. Kanalbefehlsworten besteht.

Ist im Namensfeld ein Name angegeben, so wird diesem eine implizite Länge von 1 zugewiesen.

20.18. Die Zerlegung von Programmen

Die Diskussion über Programmabschnitte [1] und gemeinsame Abschnitte [2], die wir im Abschnitt 10.4. angestrengt haben, sowie die Diskussion über Quellen- und Objektmoduln, die im Abschnitt 10.5. erfolgte, betrifft nicht nur Programme, die in der AL geschrieben werden, sondern auch solche, die in problemorientierten Sprachen abgefaßt sind. Die Eingabe für den Binder kann sowohl einen Programmabschnitt ohne Namen umfassen, der deswegen mitunter als unbenannter Programmabschnitt bezeichnet wird, aber auch benannte Programmabschnitte. In Abwesenheit von irgendwelchen Statements, die einen Abschnittsnamen festlegen, nimmt nämlich der Assemblierer die Ausgabe eines einzigen unbenannten Programmabschnittes vor.

Eine zur Übersetzung gelangende Programmeinheit kann mehr als einen Abschnitt umfassen. Zur Markierung des Beginns eines Programmabschnittes dient die Vereinbarung

```
name      CSECT    leer
```

Hierbei ist unter name der Name des beginnenden Programmabschnittes aufzuführen. Wird name weggelassen, so wird damit der unbenannte Programmabschnitt gestartet. Der Programmierer darf die Niederschrift eines Programmabschnittes unterbrechen und zu einem späteren Zeitpunkt wieder aufnehmen, da vom Assemblierer für jeden Programmabschnitt ein eigener Zuordnungszähler geführt wird. Jeder Teil eines auf diese

1) Im angloamerikanischen Sprachraum als control sections bezeichnet.
2) Im angloamerikanischen Sprachraum als common sections bezeichnet.

Weise unterbrochenen Programmabschnittes muß dann notwendig mit dem CSECT-Statement beginnen, das für diesen Abschnitt eingeführt wurde. Der Assemblierer sondert während des Übersetzungsprozesses die einzelnen Programmabschnitte von einander ab und übersetzt einen nach dem anderen. Dabei wird jeder Abschnitt grundsätzlich so übersetzt, als ob er in einem Stück vorliegen würde. Unbeachtet bleibt die tatsächliche Anzahl von Teilen, die zu jedem Abschnitt gehören.

Eine Vereinbarung, die seit Einsatz des Systems IBM /370 kaum noch benutzt wird, ist das Statement

```
name    START   leer
```

oder

```
name    START   adresse
```

Dieses Statement kann nur vor einer Instruktion oder vor einer Vereinbarung auftreten, die Einfluß auf den Wert des Zuordnungszählers haben. Unter name ist der Name des mit dieser Vereinbarung beginnenden Programmabschnittes niederzuschreiben. Unter adresse muß ein Direktwert aufgeführt sein. Der Assemblierer weist diesen Wert dem Zuordnungszähler zu. Der Hauptzweck des START-Statements ist darin zu sehen, daß man eine Adresse einführen kann, mit der die vom Assemblierer ausgedruckten Programmauflistungen beginnen, sicher besonders dann eine nützliche Sache, wenn man von vorausgehenden Binderläufen den Verschiebungsfaktor der betreffenden Programmeinheit kennt. Man erspart sich damit bei der Fehlersuche die lästigen sedezimalen Additionen. - Der angegebene Direktwert für adresse sollte durch 8 teilbar sein, anderenfalls legt der Assemblierer den Anfang des Programmes auf die nächsthöhere durch 8 teilbare Adresse. Damit ist beispielsweise 1802 gleichwertig 1808.

Fehlt der Operand, so entspricht das START-Statement dem CSECT-Statement, d.h. zur Fortsetzung eines mit der Vereinbarung START eingeleiteten Programmabschnittes genügt ein ordinäres CSECT-Statement.

Bleibt bei der Codierung des START-Statements das Namensfeld leer, so haben wir es mit dem unbenannten Programmabschnitt zu tun.

Aus den bisherigen Erörterungen ist ersichtlich, daß das START-Statement einfach als bloße Kombination einer ORG- und einer CSECT-Vereinbarung angesehen werden kann. Somit sind

```
ABC     START 8192
```

und

```
        ORG   8192
ABC     CSECT
```

gleichwertig.

Ein gemeinsamer Abschnitt muß mit dem Statement

```
name    COM   leer
```

begonnen werden. Durch name wird wie üblich der Name des Abschnitts eingeführt. Fehlt der Name, so wird der unbenannte gemeinsame Abschnitt eingerichtet. Wie die Programmabschnitte können auch gemeinsame Abschnitte unterbrochen und fortgesetzt werden, wozu die bloße Wiederholung des COM-Statements für den betreffenden Abschnitt genügt [1].

---

1) Im Betriebssystem DOS kann nur der unbenannte gemeinsame Abschnitt eingerichtet werden, d.h. das Namensfeld muß leer bleiben. Der Hinweis ist vielleicht recht interessant, daß diese Einschränkung nur für den Assemblierer gilt. Einige Kompilierer können durchaus auch im DOS benannte gemeinsame Abschnitte generieren.

Das Auftreten des gleichen Namens bei mehr als einem der Statements CSECT, START und COM bedeutet nicht die mehrfache Definition des betreffenden Namens. Wir haben es hier mit einer Ausnahme der allgemeinen Regel zu tun. Jeder Abschnitt, der einmal begonnen ist, wird solange weitergeführt, bis entweder ein neuer Abschnitt gestartet wird oder bis das Programmende festgestellt wird.

Pseudoabschnitte [1] werden durch die folgende Vereinbarung eingerichtet:

```
name      DSECT    leer
```

Durch dieses Statement wird ein in der AL geschriebener Abschnitt eingeleitet, der vom Assemblierer genau so wie jeder andere Programmabschnitt bearbeitet wird; für Pseudoabschnitte werden allerdings keine Speicherplätze reserviert. Dem Programmierer ist durch die Codierung von Pseudoabschnitten ein Mittel in die Hand gegeben, durch das er ein Hauptspeichergebiet nur einmal zu definieren braucht. Später kann er dann dieses Gebiet so oft wie nötig durch eine Bezugnahme auf es in sein Gesamtprogramm einbauen. Die DSECT-Statements müssen mit einem Namen versehen sein. [1] Der für ein DSECT-Statement eingerichtete Zuordnungszähler wird stets relativ zu Null orientiert.

Um sich auf die in einem Pseudoabschnitt vorgenommene Codierung beziehen zu können, muß der Programmierer mit einem Basisregister arbeiten:

```
USING x,register
```

Hierbei bedeutet x der Name des Pseudoabschnittes. Selbstverständlich muß der Programmierer auch dafür Sorge tragen, daß das Basisregister tatsächlich mit dem korrekten Wert geladen wird.

Um die Verwendung des DSECT-Statements zu veranschaulichen, wollen wir ein kleines Beispiel verfolgen. Wir wollen annehmen, daß bei der Verarbeitung einer Datei abwechselnd zwei völlig gleiche Eingabegebiete benutzt werden sollen. Irgendein Programmabschnitt, wir wollen hier das Hauptprogramm voraussetzen, wird dann die Definition der beiden Gebiete enthalten (siehe Abb. 20.14.).

```
HAUPT     CSECT
          .
          .
          .
EGEB1     DS     CL8Ø   ERSTES EINGABEGEBIET
EGEB2     DS     CL8Ø   ZWEITES EINGABEGEBIET
          .
          .
          .
AB        DSECT
FLD1      DS     CL4
FLD2      DS     CL17
FLD3      DS     CL41
FLD4      DS     CL18
          .
          .
          .
```

Abb. 20.14. Beispiel für einen Pseudoabschnitt in einem Programm

1) Im angloamerikanischen Sprachraum als dummy sections bezeichnet.

Wollen wir uns im Hauptprogramm HAUPT auf die verschiedenen Felder eines Eingabegebietes beziehen, so müssen wir zunächst einmal sagen, mit welchem Register wir die Eingabegebiete ansprechen wollen. Soll die Referenz beispielsweise über das Register 3 erfolgen, so richten wir als Basisregister mittels der Vereinbarung

```
USING AB,3
```

ein. Bevor wir uns nun tatsächlich auf die Eingabegebiete beziehen können, müssen wir das Basisregister 3 laden, entweder also durch die Instruktion

```
L       3,=A(EGEB1)
```

oder durch die Instruktion

```
L       3,=A(EGEB2)
```

Welche Instruktion im Einzelfall auszuführen ist, hängt davon ab, welches Eingabegebiet (EGEB1 oder EGEB2) an der Reihe ist.

Da weder Konstanten noch Instruktionen eines Pseudoabschnittes übersetzt werden, wird normalerweise das DS-Statement das einzige Statement sein, das in einem Pseudoabschnitt codiert wird. Innerhalb eines Pseudoabschnittes definierte Namen dürfen in Adreßkonstanten nur dann benutzt werden, wenn jeder solche Name mit einem anderen Namen aus dem gleichen Pseudoabschnitt gepaart vorkommt, wobei die beiden Namen zusätzlich entgegengesetzte Vorzeichen aufweisen müssen.

In der Symboltabelle werden die Namen aller Abschnitte mit einer impliziten Länge von einem Byte ausgestattet.

In vielen Fällen bewirkt ein gezielter Einsatz von Pseudoabschnitten eine Vereinfachung der Programmierung.

Sehen wir den Literalpool (siehe Abschnitt 20.17.) noch unter einem anderen Aspekt. Das Auftauchen eines END-Statements bedeutet gleichzeitig implizit auch das Vorhandensein des LTORG-Statements. Alle bis dahin vorkommenden Literale gehören dann zum gerade übersetzten Programmabschnitt. Will der Programmierer die Literale an andere Stellen des Programmes legen, so muß er eigene LTORG-Statements codieren.

Man sollte mit Nachdruck betonen, daß die meisten kleineren Programme nur aus einem Programmabschnitt bestehen sollten und daß es unnötig ist, diesen zu benennen. Die in diesem Abschnitt beschriebenen Sprachmittel erfüllen ihren Zweck hauptsächlich bei umfangreicheren Programmen.

20.19. Externe Namen

Die meisten der Namen, die in einem in der AL geschriebenen Programm verwendet werden, sind interne Namen, d.h. sie sind nur innerhalb der jeweiligen Übersetzung bekannt. Einige wenige Namen werden als externe Namen eingeführt, d.h. sie werden auch dem Binder (Linkage Editor) und anderen Programmen bekanntgemacht, die sich des betreffenden Objektmoduls bedienen müssen. Die Namen benannter gemeinsamer Abschnitte und von benannten Programmabschnitten gelten automatisch als externe Namen. Die Namen anderer Stellen eines Programms können ebenfalls zu externen Namen gemacht werden. Zu diesem Zweck ist das Statement

```
leer     ENTRY   name1,name2,···,namen
```

zu benutzen. Es macht die unter $name_1$, $name_2$, ···, $name_n$ aufgeführten Namen außerhalb des Programmes bekannt. Sie gelten dann als sogenannte Eingangsnamen. Jeder Operand muß hierbei der Name eines verschieblichen Symbols sein, das nicht in einem Pseudoabschnitt definiert sein darf.

In einem Programm kann durchaus auch die Erfordernis vorliegen, daß es sich auf Plätze bezieht, die in andere Programme eingebettet sind. Für diese Zwecke können wir eines der beiden nachfolgenden Statements benutzen:

leer	EXTRN	$name_1$,$name_2$,···,$name_n$
leer	WXTRN	$name_1$,$name_2$,···,$name_n$

Diese beiden Statements teilen dem Assemblierer mit, daß Bezugnahme auf die Symbole mit den Namen $name_1$, $name_2$ usw. nicht im Programm selbst aufgelöst werden sollen. Der Assemblierer ist viermehr angehalten, den Binder darüber zu informieren, daß diese Namen sich auf Eingangsnamen anderer Abschnitte beziehen. Der Unterschied zwischen den Statements EXTRN und WXTRN ist folgender: Bei einem durch EXTRN erklärten externen Symbol wird der Binder veranlaßt, zunächst die ihm zum Binden übergebenen Programmbausteine nach dem Vorhandensein des betreffenden Symbols zu überprüfen. Wird das Symbol dabei nicht gefunden, so soll die Suche nach dem externen Symbol in einer Bibliotheksdatenmenge fortgesetzt werden, um evtl. Abschnitte zu finden, in denen diese externen Symbole auftreten. Erst nach vergeblicher Suche in dieser Bibliothek gilt das externe Symbol als nicht aufgelöst. Bei einem schwachen externen Symbol (Statement WXTRN) wird die automatische Durchsuchung einer Bibliothek nicht durchgeführt, solche Symbole gelten also bereits vorzeitig als nicht aufgelöst. Jeder Versuch in einem Programm, sich auf ein nicht aufgelöstes Symbol zu beziehen, ist zwar akzeptabel, doch der Inhalt des betreffenden Speicherplatzes wird ungültig sein und bei Benutzung als Basisregister sogar meist zu unerwarteten Programmunterbrechungen führen.

Eine Adreßkonstante vom V-Typ ist gleichwertig einer Adreßkonstante des A-Typs bei gleichzeitiger Eintragung ihres Namens in ein EXTRN-Statement. Man kann also sowohl

```
EXTRN EEHH
DC    A(EEHH)
```

als auch

```
DC    V(EEHH)
```

codieren. Erscheint ein Name zugleich in einem EXTRN- und in einem WXTRN-Statement, so wird nur sein erstes Auftreten verarbeitet. Tritt ein Name zugleich in einer Adreßkonstanten vom V-Typ und in einem WXTRN-Statement auf, so wird das Symbol als zum WXTRN-Statement zugehörig betrachtet.

Die Benutzung von übermäßig vielen Symbolen im ENTRY- und EXTRN- (bzw. WXTRN-) Vereinbarungen ist nicht anzuraten, erschwert sie doch das Durchleuchten von Programmen in außerordentlicher Weise. Da die Namen von Programmabschnitten und von gemeinsamen Abschnitten gleichzeitig als Eingangsnamen gelten , brauchen sie nicht in ENTRY-Statements einzugehen. Auf der anderen Seite werden Bezugnahmen auf andere Abschnitte gewöhnlich durch Adreßkonstanten des V-Typs praktiziert, die wiederum EXTRN- und WXTRN-Statements überflüssig machen.

20.20. Rettungsgebiete (Sicherungsgebiete)

Wenn ein Unterprogramm aufgerufen wird, so beachten beide Betriebssysteme (DOS und OS) gewissen Konventionen. Diese wollen wir jetzt kennenlernen.

Vor dem Ansprung des Unterprogrammes wird die Adresse der ersten Instruktion des Unterprogrammes ins allgemeine Register 15 gestellt. Die Adresse, zu der das Unterprogramm nach seiner Beendigung zurückkehren soll, wird ins allgemeine Register 14 geladen. Außerdem verweist das Register 13 auf ein Rettungsgebiet [1], das in der aufrufenden Programmeinheit installiert ist. Das Rettungsgebiet ist mindestens 18 Vollwörter (gleich 72 Bytes) lang. In diesem Abschnitt wollen wir es detailliert besprechen.

Wort	*Distanz*	*Bedeutung*
1	0	Benutzung durch PL/1 zur Kennzeichnung der Bearbeitung der Unterbrechungen und zur Festhaltung der Länge dieses Rettungsgebietes
2	4	Adresse des Rettungsgebietes, das von der rufenden Programmeinheit benutzt wird
3	8	Adresse des Rettungsgebietes, das in der aufgerufenen Programmeinheit bereitgestellt ist
4	12	Inhalt des Registers 14 beim Eintritt in die aufgerufene Programmeinheit (Rückkehradresse)
5	16	Inhalt des Registers 15 beim Eintritt in die aufgerufene Programmeinheit (Adresse der Eingangsstelle der aufgerufenen Programmeinheit)
6	20	Inhalt des Registers 0
7	24	Inhalt des Registers 1
8	28	Inhalt des Registers 2
9	32	Inhalt des Registers 3
10	36	Inhalt des Registers 4
11	40	Inhalt des Registers 5
12	44	Inhalt des Registers 6
13	48	Inhalt des Registers 7
14	52	Inhalt des Registers 8
15	56	Inhalt des Registers 9
16	60	Inhalt des Registers 10
17	64	Inhalt des Registers 11
18	68	Inhalt des Registers 12

Abb. 20.15. Aufbau der ersten 18 Wörter der Rettungsgebiete (Sicherungsgebiete) von Programmeinheiten

1) Im angloamerikanischen Sprachraum als Save Area bezeichnet.

Abb. 20.15. zeigt uns den Inhalt der einzelnen Wörter des Rettungsgebietes auf. Das erste Wort braucht im allgemeinen von einem Programmierer nicht benutzt zu werden, wenn er sich der AL bedient. Eine Ausnahme hiervon ist jedoch dann zu machen, wenn sich das aufgerufene Unterprogramm auf in PL/1 geschriebene Routinen stützt. Die Versorgung des zweiten Wortes wird von der jeweiligen Programmeinheit selbst besorgt, die des dritten Wortes wird von der aufgerufenen Programmeinheit erledigt. Der Inhalt des zweiten Wortes ist für den Fall von Bedeutung, daß eine Programmeinheit fehlerhaft zu Ende geht. Man kann aus ihm ersehen, wie die einzelnen Programmeinheiten miteinander verkettet sind, d.h. die Aufrufkette zurückverfolgen. Die restlichen Wörter werden benutzt, um den Inhalt der Register 14 bis 12 wegzuspeichern, den sie zum Zeitpunkt des Aufrufs besaßen, d.h. in dem Augenblick, in dem die Programmablaufsteuerung an das aufgerufene Unterprogramm überging. In der aufgerufenen Programmeinheit muß ebenfalls ein Rettungsgebiet eingerichtet sein, in die sie selbst, wie bereits bekannt, den Inhalt des Registers 13 wegspeichern muß, wenn sie die Programmablaufsteuerung übernimmt. Der Inhalt des Registers 13 wird dabei, wir sagten es schon, ins zweite Wort des Rettungsgebietes abgelegt.

Normal werden die soeben geschilderten Vorgänge meist so codiert, wie es die Abb. 20.16. zeigt. Aus der gleichen Abbildung können wir ersehen, wie die Codierung aussehen muß, um die Rückkehr in die aufrufende Programmeinheit zu gewährleisten.

Bei der Vorbereitung des Rücksprungs ist folgendes zu beachten: Wenn kein Rückkehrcode an die aufrufende Programmeinheit zurückgegeben werden soll, kann selbstverständlich die LA-Instruktion zum Laden des Rückkehrcodes entfallen. Natürlich kann auch die MVI-Instruktion weggelassen werden, die das erste Byte des vierten Wortes des Rettungsgebietes auf sedezimal FF setzt, d.h. alle Bitstellen auf 1 bringt. Doch ein solches Markieren von benutzten Rettungsgebieten kann zu einer äußerst wertvollen Hilfe werden, wenn Speicherabzüge (Dumps) vorliegen, die nach aufgetretenen Programmfehlern produziert wurden.

Im Betriebssystem OS werden die Steuerprogramme mit einem Hauptprogramm in genau der gleichen Weise verknüpft wie ein Hauptprogramm mit einem Unterprogramm. Deshalb sollte die in Abb. 20.16. gezeigte Codierung oder zumindest eine funktionell gleiche jedes Hauptprogramm einleiten und beenden. Innerhalb des Betriebssystems OS werden diese Beziehungen jedenfalls strikt eingehalten. Es fehlt allerdings bei der Rückkehr zu aufrufenden Programmeinheiten die Instruktion MVI.

Im Betriebssystem DOS liegen die Dinge etwas anders. Die Programmablaufsteuerung wird vom Betriebssystem ohne spezielle Vorkehrungen an ein Hauptprogramm übergeben, das selbst wiederum mittels einer SVC-Instruktion zum Betriebssystem zurückkehrt. Die hier gezeigte Verbindungstechnik wird nur für Unterprogramme benutzt.

Es ist nicht notwendig, daß in einer aufgerufenen Programmeinheit ein Rettungsgebiet eingerichtet wird, es sei denn, daß sie selbst wiederum eine andere Programmeinheit aufruft. Im Betriebssystem OS jedoch erfordern die Makros für die Eingabe und Ausgabe die Installation eines Rettungsgebietes, da sie selbst eine Verbindung zu den passenden EA-Routinen herstellen. Im Betriebssystem DOS bedeutet die Verwendung einiger spezieller Makros ebenfalls die Notwendigkeit, ein Rettungsgebiet einzurichten, doch bei den am häufigsten verwendeten Makros kann darauf verzichtet werden.

Da das allgemeine Register 13 die Adresse des Rettungsgebietes enthält, das für das betreffende Unterprogramm angelegt ist, kann es zur Zeit der Unterprogrammausführung durchaus auch als Basisregister eingesetzt werden. Durch eine simple Erweiterung dieses Gebietes kann zusätzlich auch Platz für Arbeitsdaten (für die Variablen und für die Konstanten) geschaffen werden. Bei umfangreicheren Programmen kommt es oft zu einem Engpaß an Basisregistern; diese Technik rettet dem Anwender ein Register.

Codierung der aufgerufenen Programmeinheit

```
AGPE     CSECT
         USING *,y
         STM   14,12,12(13)
         LR    y,15
         LA    z,RETGEB  LADEN DER ADRESSE DES RETTUNGSGEBIETES
         ST    z,8(13)
         ST    13,4(z)
         LR    13,z
         :
RETGEB   DS    18F       RETTUNGSGEBIET
         :
*  RUECKKEHR IN DIE AUFRUFENDE PROGRAMMEINHEIT
*    - VORBEREITUNG UND AUSFUEHRUNG -
         L     13,4(13)
         LM    14,12,12(13)
         MVI   12(13),X'FF'
         LA    15,c
         BR    14
         :
         END
```

Es bedeuten:

y, z	Beliebige allgemeine Register mit Ausnahme der Register 13, 14 und 15
c	Rückkehrcode

Abb. 20.16. Ansprechen der Rettungsgebiete von aufrufender und aufgerufener Programmeinheit in der aufgerufenen Programmeinheit

20.21. Der Austausch von Daten zwischen Programmeinheiten

Für die Übergabe von Daten an eine aufgerufene Programmeinheit und für die Übernahme von Daten aus einer solchen sollte eine Standardmethode eingesetzt werden. Die hierzu verwendete Codierung sollte der ähneln, die in Abb. 20.17. demonstriert ist.

Codierung in der aufrufenden Programmeinheit

```
            .
            .
            .
            L       15,=V(eingangsadresse)
            L       1,=A(aliste)
            BALR    14,15
ra          . . .
            .
            .
            .
aliste      DC      A(arg1)
            DC      A(arg2)
            .
            .
            .
            DC      X'8Ø',AL3(argn)
            .
            .
            .
```

Es bedeuten:

eingangsadresse	Adresse der Eingangsstelle der aufgerufenen Programmeinheit
aliste	Argumentenliste
argi	Argument$_i$
ra	Rückkehrstelle nach Ablauf der aufgerufenen Programmeinheit

Abb. 20.17. Standardmethoden des Austausches von Dateien zwischen Programmeinheiten (aufrufende Programmeinheit)

Man beachte hierbei, daß das allgemeine Register 1 vor dem Aufruf des Unterprogrammes mit der Adresse der Argumentenliste geladen wird. Sind von der aufrufenden Programmeinheit keine Argumente an die aufgerufene Programmeinheit zu übergeben, so sollte Register 1, d.h. das Standardverbindungsregister, auf Null gesetzt werden, um klar und deutlich das Fehlen der Argumentenliste zu signalisieren.

In der aufgerufenen Programmeinheit, d.h. im Unterprogramm, sind natürlich Vorkehrungen zu treffen, um die Argumente übernehmen zu können, d.h. es ist parameterabhängig zu machen. An Stelle eines Parameters tritt dann das entsprechende Argument aus der aufrufenden Programmeinheit. Auch hier empfiehlt sich eine standardisierte Programmierung, so wie sie die Abb. 20.18. zeigt.

Codierung in der aufgerufenen Programmeinheit

```
        .
        .
        .
L       w,Ø(1)
L       x1,Ø(w)
L       x2,4(w)
L       x3,8(w)
        .
        .
        .
```

Es bedeuten:

w	Allgemeines Register zur Aufnahme der Adresse der Argumentenliste der aufrufenden Programmeinheit
x_i	Allgemeines Register zur Aufnahme der Adresse des Argumentes i aus der aufrufenden Programmeinheit

Abb. 20.18. Standardmethoden des Austausches von Daten zwischen Programmeinheiten (aufgerufene Programmeinheit)

Die in Abb. 20.18. dargestellte Codierung sorgt dafür, daß im allgemeinen Register x_1 die Adresse des ersten Argumentes, im Register x_2 die des zweiten Argumentes usw. steht.

Es ist eine Übereinkunft darüber erzielt worden, daß die linksbündige Bitstelle des letzten Wortes der Argumentenliste (siehe Abb. 20.17.) auf 1 gesetzt wird. Auf diese Weise kann im Unterprogramm gezielt abgefragt werden, ob das letzte Argument bereits gefunden ist oder nicht. Gestattet die Logik des Unterprogrammes eine exakte Auskunft darüber, wieviel Argumente übernommen werden müssen, so kann man sich diese Technik sparen.

Wenn im Betriebssystem OS das EXEC-Statement der Jobsteuersprache eine PARM-Angabe mit einem Wert aufweist, so besteht die ins Hauptprogramm transferierte Argumentenliste aus nur einem Wert. Dieses Wort wiederum enthält die Adresse des Argumentes. In diesem Fall besteht das Argument insgesamt aus einem Halbwort, in dem die Länge des Argumentes vermerkt ist und dem Argument selbst, das eine Länge zwischen 0 und 100 Bytes besitzen kann.

Funktionsunterprogramme in der FORTRAN-Sprache geben in der Regel einen Funktionswert an die aufrufende Programmeinheit zurück. Der Funktionswert wird dabei ins allgemeine Register 0 hineingestellt, wenn er in Festpunktdarstellung vorliegt, dagegen ins Gleitpunktregister 0, wenn er ein Gleitpunktwert ist. Viele von der IBM herausgegebene Makros geben Werte in den allgemeinen Registern 0 und 1 zurück. Bei der Benutzung von Makros sollte man grundsätzlich immer damit rechnen, daß durch sie die Werte in den Registern 14, 15, 0 und 1 geändert werden. Von den anderen Registern hingegen kann erwartet werden, daß Makros die in ihnen enthaltenen Werte unberührt lassen.

Aufgabengruppe 56

1) Welcher Irrtum ist vom Programmierer bei der Codierung der nachfolgenden Statements begangen worden (sofern überhaupt ein Irrtum vorliegt)?

```
a) ALPHA    DS    C8
b) 2BETA    AP    X,Y+2
c)          A     X,X
d) GAMMA    DS    F'18ØØØ'
e)          M     7,7*GAMMA+413
f) DELTA    DR    7,2
g) SPIL 4   AW    2,XYR
```

2) Schreibe unter Benutzung der Makros von Abschnitt 4.5. ein Programm, das den Inhalt einer Lochkarte einliest, die wie folgt abgelocht ist:

Spalten 1 bis 4: x (vierstellige ganze Zahl)
Spalten 5 bis 9: y (fünfstellige ganze Zahl)

Im Programm sind die Werte von z zu berechnen und zusammen mit den x- und y-Werten auf eine Zeile zu drucken. Die z-Werte sollen gemäß der beiden folgenden Formeln bestimmt werden:

$z = \sqrt{x^2+z^2}$ für $y>7x-4$

$z = x+y$ für $y \leqq 7x-4$

Alle z-Werte sollen exakt bis auf ± 1 berechnet werden. Verwende dabei das Programm von Aufgabe 4) der Aufgabengruppe 54 als getrennten Programmabschnitt.

3) Ändere die Aufgabe 2) so ab, daß z für alle x- bzw. y-Werte bis auf zwei Dezimalstellen genau berechnet werden kann!

4) Es sollen in einem Programmabschnitt die Inhalte von 80spaltigen Lochkarten verarbeitet werden, die wie folgt eingeteilt sind:

Spalten 1 bis 10: Feld 1
Spalten 11 bis 20: Feld 2
Spalten 21 bis 30: Feld 3
...
Spalten 71 bis 80: Feld 8

Jedes Feld wiederum ist wie folgt unterteilt: Auf seinen ersten 5 Spalten soll ein x-Wert, auf den nächsten 4 Spalten der zugehörige y-Wert erscheinen. Die letzte Spalte jedes Feldes soll leer bleiben. Codiere zunächst einen Pseudoabschnitt, der die Feldeinteilung beschreibt. Anschließend ist Aufgabe 2) so zu modifizieren, daß Karten mit dieser Einteilung eingegeben werden können. Für jede eingelesene Karte sind acht Druckzeilen auszugeben.

5) Eine zweistellige Zahl ist im Feld X als gepackte Dezimalzahl gespeichert; das mit X benannte Feld belegt drei Bytes. Zu dieser zweistelligen Zahl soll eine im Feld Y gespeicherte vierstellige Zahl addiert und die Summe im Feld Z gespeichert werden. Sowohl Y als auch Z sind drei Bytes groß; beide Felder sind für die Speicherung gepackter Dezimalzahlen vorgesehen. Wenn die Summe aus X und Y eine fünfstellige Zahl ergibt, so sollen nur die ersten vier Ziffern dieser Zahl im Feld Z gespeichert werden. Schreibe ein Programm in der AL, das diese Rechnungen ausführt und anschliessend zum Speicherplatz verzweigt, dem der symbolische Name PSTT gegeben wurde.

6) In einem Programmabschnitt sind für A, B, C, D, E und F Gleitpunktzahlen einfacher Genauigkeit einzulesen. Anschließend sind die Lösungen des linearen Gleichungssystems

$$\begin{aligned} \text{I.} \quad & Ax + By = C \\ \text{II.} \quad & Dx + Ey = F \end{aligned}$$

zu berechnen; x und y sollen dabei ebenfalls Gleitpunktzahlen einfacher Genauigkeit sein. Wenn AE=BD ist, so existieren keine Lösungen, es sei denn, daß gleichzeitig auch AF=CD ist; in diesem Fall ist eine unendliche Zahl von Lösungen vorhanden. Bei den beiden letzten Möglichkeiten soll x=y=0 gesetzt werden, ansonsten sind die eindeutigen Lösungen auszugeben.

7) Schreibe einen Programmabschnitt, der die Inhalte von Lochkarten einliest, auf denen je sechs zehnstellige ganze Zahlen abgelocht sind. Jede dieser Zahlen soll in eine normalisierte Gleitpunktzahl einfacher Genauigkeit umgewandelt werden und auf die Speicherplätze gespeichert werden, denen die symbolischen Adressen A, B, C, D, E und F zugeordnet sind.

8) Unter Benutzung der Ergebnisse der Aufgabe 5) von Aufgabengruppe 55 ist ein Programmabschnitt zu schreiben, der eine Zeile druckt, die in acht zehnstellige Felder ohne Dezimalpunkt eingeteilt ist (aus Gründen der Lesbarkeit sind zwischen den Feldern Leerstellen zu lassen).

Anschließend sind folgende weiteren Programmabschnitte zu verfassen:

a) Lesen der Inhalte von Lochkarten, deren Einteilung gleich der der in Aufgabe 7 dieser Aufgabengruppe beschriebenen Karten ist.
b) Bestimmung der Lösungen der Gleichungen gemäß der Aufgabe 6 dieser Aufgabengruppe.
c) Aufbereitung der Lösungen zum Drucken auf der Basis des in Aufgabe 6 der Aufgabengruppe 55 geschilderten Sachverhalts, d.h. unter Benutzung des dortigen Unterprogramms.

Alle Programmabschnitte sind zu einem Programm zusammenzufügen.

9) Beginnend beim Speicherplatz mit der symbolischen Adresse G ist eine Folge von 135 ganzen Dualzahlen in je einem Wort in aufsteigender Reihenfolge gespeichert. Beispiel:

```
G     DC     F'135'
      DC     F'138'
      ...
      DC     F'998764'
      DC     X'FFFFFFFF'
```

Es ist ein Programmabschnitt zu schreiben, mit dessen Hilfe es möglich ist, die Tabelle zu durchsuchen. Das Suchargument K stehe im Register 2, es soll, beginnend beim Tabellenanfang, solange mit den Tabellenwerten verglichen werden, bis ein Tabellenwert gefunden wird, der nicht mehr größer als das Suchargument ist. Ins Register 1 ist die Adresse des größten Tabellenwertes zu stellen, der noch kleiner als das Suchargument ist.

10) Es ist ein neues Programm für das Tabellensuchen zu entwickeln. Das Suchen soll dieses Mal in der Tabellenmitte beginnen, d.h. der erste Vergleich zwischen K und einem Tabellenwert soll mit dem Tabellenwert stattfinden, der genau in der Mitte der Tabelle gespeichert ist. Dadurch kann bestimmt werden, in welcher Tabellenhälfte

das Suchen fortzusetzen ist. Die nächste Prüfung soll mit dem mittleren Tabellenwert der ausgewählten Hälfte erfolgen. Dieses Verfahren soll solange weitergeführt werden, bis das gewünschte Ergebnis erhalten wird. Vor der Codierung soll zunächst einmal ein Programmablaufplan gezeichnet werden.

Das hier beschriebene Verfahren ist als binäres Tabellensuchen bekannt. Es erfordert ein wenig mehr Programmieranstrengungen als das lineare Tabellensuchen (Aufgabe 9). Beim Durchsuchen umfangreicher Tabellen ergeben sich jedoch durch das binäre Tabellensuchen recht beachtliche Zeiteinsparungen.

21. Makrostatements für das Betriebssystem DOS

21.1. Die Programmierung der Eingabe und Ausgabe

Die Programmierung der Eingabe/Ausgabe-Operationen ist kompliziert und aufwendig zugleich. Das zeigt sich besonders deutlich dann, wenn man an die Programmierung der Fehlerbehandlung herangeht. Als bemerkenswerter Vorteil erweist es sich jedoch, daß die Programmierung der Eingabe/Ausgabe für die überwiegende Mehrzahl der Programme durch beinahe gleiche Instruktions- und Befehlsfolgen erfaßt werden kann. Daraus folgt augenscheinlich, daß der Arbeitsaufwand des Programmierers beträchtlich verringert werden kann, wenn standardisierte Eingabe/Ausgabe-Routinen zur Ausführung aller EA-Operationen benutzt werden können. Um eine solche standardisierte Routine in ein beliebiges Programm einbauen zu können, muß dreierlei beachtet werden:

1) Eine geeignete Routine muß in der verschieblichen Programmbibliothek verfügbar sein.

2) Der Programmierer muß nach einer vorgeschriebenen Methode die Eigenschaften der Datenmenge angeben können, so daß das Übersetzungsprogramm exakt weiß, welche Routine zu verwenden ist, welche Parameterwerte zuzuweisen sind usw.

3) Wenn eine EA-Operation ausgeführt werden soll, muß der Programmierer an Stelle eines mnemonischen Operationsschlüssels einen Pseudocode niederschreiben. Wenn das Übersetzungsprogramm diesen Code erkannt hat, ersetzt es ihn durch die Standardstatements, die die gewünschte Operation ausführen. Diese sogenannten Makrocodes sind nur Spezialfälle der allgemeinen Makrosprache des Systems IBM/370, die wir später an verschiedenen Stellen kurz diskutieren wollen.

Die am häufigsten verwendeten Makroinstruktionen wollen wir zunächst einmal in einer Übersicht festhalten:

a) OPEN

Durch diese Makroinstruktion werden Band/Platten-Kennsätze gelesen und geprüft oder geschrieben (Vorsätze)

b) CLOSE

Durch diese Makroinstruktion werden die Nachsätze genau so behandelt, wie durch OPEN die Vorsätze behandelt werden.

c) GET

Durch diese Makroinstruktion wird der nächste logische Satz dem Programm zur Verfügung gestellt.

d) PUT

Diese Makroinstruktion bewirkt, daß der angegebene logische Satz für die Ausgabe bereitgestellt wird.

Bei den EA-Routinen, die wir hier behandeln wollen, braucht der Programmierer nicht die Kanalbefehlswörter niederzuschreiben, da diese vom IOCS (Input/Output Control System - Eingabe/Ausgabe Steuerungssystem) bereitgestellt werden. Diese Routinen handhaben auch das Blocken und Entblocken von Sätzen, sie behandeln ferner auftretende Fehlerbedingungen. Faßt man alle Aspekte zusammen, so kann man wohl sagen, daß sie

im allgemeinen dem Programmierer die Freiheit geben, sich ganz auf den logischen Ablauf seines Programmes zu konzentrieren; die Einzelheiten der Eingabe/Ausgabe brauchen ihn nicht mehr zu interessieren. Die hier erwähnten Routinen sind unter dem Begriff „Logisches IOCS" klassifiziert. Neben diesen Routinen gibt es auch noch einen anderen Typ von IOCS-Routinen; diese sind unter dem Begriff „Physisches IOCS" bekannt. Beim physischen IOCS muß der Programmierer die Kanalbefehlswörter (CCW) selbst schreiben. Wenn nun eine EA-Operation ablaufen soll, gibt er eine Makroinstruktion an, die die Ausführung eines eigenen Kanalprogrammes bewirkt. Die Routinen des physischen IOCS bauen das Kanaladreßwort (CAW) auf und behandeln, falls gewünscht, Fehlerbedingungen und andere ungewöhnliche Bedingungen. Sie verarbeiten sogar Kennsätze. Der Programmierer behält jedoch die Verantwortung über die detaillierte Verarbeitung aller eingegebenen und auszugebenden Informationen. Das physische IOCS erfordert auf Seiten des Programmierers wesentlich größere Anstrengungen als das logische IOCS. Es gestattet ihm jedoch, Verfahren zu handhaben, die das logische IOCS nicht behandeln kann. Beispielsweise kann er Datenmengen auf solchen externen Speichergeräten ansprechen, für die noch keine Routinen des logischen IOCS zur Verfügung stehen, er kann ferner eine besondere Behandlung von Fehlern und ungewöhnlichen Bedingungen vorsehen usw. Beim System IBM/370 können sowohl das logische IOCS als auch das physische IOCS benutzt werden. Zuerst wollen wir das logische IOCS behandeln.

21.2. Dateidefinitionen

Damit für eine zu lesende bzw. zu schreibende Datei die geeigneten IOCS-Routinen erzeugt bzw. in das Programm eingefügt werden können, muß die Datei definiert werden. Das geschieht durch das Makrostatement zur Definition von Dateien (Define The File, abgekürzt DTF). Betrachten wir dazu ein erstes Beispiel (Abb. 21.1.).

```
KTEIN    DTFCD DEVICE=2540,BLKSIZE=80,                    1 →
               WORKA=YES,DEVADDR=SYSIPT,                  2 →
               IOAREA1=EGEB,TYPEFLE=INPUT,                3 →
               RECFORM=FIXUNB,                            4 →
               EOFADDR=DATENDE
```

Anmerkung: Die Zahlen rechts sollen anzeigen, daß Fortsetzungszeilen folgen (in der Regel werden sie auf Spalte 72 notiert)

Abb. 21.1. Beispiel einer Dateidefinition

Für jede Datei ist ein gesondertes DTF-Makrostatement zu schreiben. Aus jedem DTF-Makro wird vom Assemblierer eine Menge von Instruktionen, Konstanten und Arbeitsgebieten in Form einer Tabelle generiert. Die Ausführung des OPEN-Statements hat zur Folge, daß die Tabelle vervollständigt wird, so daß damit die Datei verarbeitet werden kann. Es wird vorgeschlagen, alle DTF-Eintragungen in einem betrimmten Programmteil zu konzentrieren. Gewöhnlich nimmt man dazu das Programmende her. Ein solches Vorgehen sichert eine größere Übersicht und damit zugleich auch eine bessere Wartbarkeit der Programme.

Die IOCS-Routine zur Verarbeitung der Datei kann als Teil der gleichen Programme übersetzt werden. Gewöhnlich wird sie jedoch in die verschiebliche Programmbibliothek

aufgenommen und steht dort zur automatischen Verbindung mit dem Programm zur Verfügung, eine bekanntlich vom Binder besorgte Handlung.

Das in Abb. 21.1. dargestellte Makrostatement DTFCD bewirkt die Definition einer Eingabedatei, die dem Kartenleser IBM 2540 mit dem logischen Einheitennamen SYSIPT zugewiesen wird. Von jeder Lochkarte werden alle 80 Spalten eingelesen. Der Programmierer hat weiterhin dafür Sorge getragen, daß zum Gebrauch für die Eingaberoutine ein 80stelliges Eingabegebiet bereitgestellt wird, das den Namen EGEB trägt. Das IOCS verwendet dieses Gebiet als Puffer. Jedes Eingabestatement bezieht sich außerdem auf ein Arbeitsgebiet, in das die Eingaberoutine den gelesenen Satz übertragen soll. Nachdem ein Satz in das Arbeitsgebiet hinein gestellt worden ist, liest die Eingaberoutine bereits den nächsten Satz nach EGEB in Voraussicht einer späteren Bezugnahme auf ihn. Die IOCS-Routine versetzt den Computer nur dann in den Wartestatus (oder überträgt beim Mehrprogrammbetrieb die Programmablaufsteuerung an eine andere Partition), wenn ein GET-Makrostatement für den nächsten Satz auf die Vollendung des Kartenlesens zu warten hat. Das Auftreten der Dateiende-Bedingung veranlaßt das IOCS, die Programmablaufsteuerung zu der Instruktion zu übertragen, die mit DATENDE markiert ist. Die normale Rückkehr von den GET-Statements zum nächstfolgenden Statement wird in diesem Fall aufgehoben.

Es gibt mehrere Arten von Dateidefinitionen, d.h. DTF-Statements. Wir werden nur einige betrachten und selbst bei diesen nicht alle Wahlmöglichkeiten (Optionen) besprechen.

Die Statements, die in der Regel auf die Datei KTEIN bezogen werden, lauten:

```
OPEN   KTEIN
GET    KTEIN,ARB
CLOSE  KTEIN
```

Hierbei stellt ARB ein 80stelliges Arbeitsgebiet dar, in dem der Programmierer die Eingabedaten anspricht und verarbeitet.

21.3. Makrostatements für die sequentielle Verarbeitung

Bevor es zu irgendeiner auf eine Datei bezogenen EA-Operation kommen kann, muß die betreffende Datei eröffnet werden. Das Eröffnen von Dateien schließt das Setzen von Bits an den dafür vorgesehenen Stellen der DTF-Tafel ein, ferner das Erzeugen oder Prüfen von Kennsätzen (Etiketten) usw. Durch das Setzen von Bits wird der Zustand der EA bei der betreffenden Datei festgehalten.

Das Makro, durch das der Eröffnungsprozeß realisiert wird, heißt allgemein

```
marke   OPEN   dateiname1,···,dateinamen
```

Die Angabe einer Marke ist freigestellt; wird sie angegeben, so bezieht sie sich auf das erste Byte, das vom Maschinencode belegt wird, der aus dem Makro generiert wird. In diesem Byte beginnt grundsätzlich eine Instruktion. Im allgemeinen ist es wirkungsvoller, wenn mit einem einzigen OPEN-Statement so viele Dateien wie möglich eröffnet werden.

Die logischen Sätze werden dem Problemprogramm durch das GET-Statement zur Verfügung gestellt, das in der Form

```
marke   GET    dateiname
```

verwendet wird, wenn auf ein Arbeitsgebiet verzichtet wird. Bei Benutzung eines Arbeitsgebietes lautet die allgemeine Form des GET-Statements wie folgt:

```
marke   GET    dateiname,arbeitsgebietname
```

Unter arbeitsgebietname ist hierbei der Name des vorgesehenen Arbeitsgebietes zu verstehen. In dieses wird der logische Satz übertragen und zur Verarbeitung bereitgestellt. Wird auf die Verwendung eines Arbeitsgebietes verzichtet, so stellt das GET-Makro die Adresse des Eingabegebietes in das allgemeine Register hinein, das im DTF-Makro für diesen Verwendungszweck definiert wurde.

Die Ausgabe von Sätzen wird durch die Statements

```
marke   PUT    dateiname
marke   PUT    dateiname,arbeitsgebietname
```

vorgenommen. Wie bei den GET-Statements ist auch bei ihnen die Angabe einer Marke freigestellt, wird kein Arbeitsgebiet angegeben, so nach Vollendung des PUT-Statement die Adresse des Ausgabegebietes in einem allgemeinen Register enthalten. Das Register wird beim OPEN-Statement eingerichtet und zur Zusammenstellung des Satzes verwendet. Jedes PUT unterstützt ein neues Ausgabegebiet, in dem der nächste Satz zusammenzustellen ist.

Das Makro CLOSE weist die Form

```
marke   CLOSE dateiname1,···,dateinamen
```

auf. Es bewirkt, daß der Datei der Verfügbarkeitsstatus entzogen wird. Es sorgt außerdem dafür, daß alle noch in Ausgabepuffern verbliebenen Sätze geschrieben werden, daß alle Nachsätze aufgezeichnet oder geprüft werden und daß die DTF-Tafel entsprechend modifiziert wird. Jede eröffnete Datei muß vor dem Jobende (Auftragsende) wieder abgeschlossen werden. Jede einmal abgeschlossene Datei kann wieder neu eröffnet werden.

Unter den sonstigen in ein Programm eingefügten Makros wird sich vielfach das Makro

```
marke   CNTRL dateiname,code,n,m
```

befinden. Dieses Makro dient dazu, die Einstellung von Magnetbändern, den Papiertransport bei Druckern und die Auswahl von Kartenablagefächern usw. zu veranlassen. In der Abb. 21.2. sind die Werte aufgeführt, die für die Operanden code, n und m niedergeschrieben werden dürfen. Sind keine Operanden aufgeführt, so dürfen auch keine im Makro hinein gestellt werden.

Bei Druckern kann geprüft werden, ob eine Lochung in den Kanälen 9 oder 12 vorhanden ist, wenn man das Makro

```
marke   PRTOV dateiname,n,routinename
```

verwendet. Unter n ist entweder 9 oder 12 anzugeben, je nachdem welche Prüfung erfolgen soll. Wird während der letzten Druckeroperation ein Papiertransport festgestellt, der über den spezifizierten Lochbandkanal hinweggleitet, so verzweigt das Programm zu der Routine, die an der durch routinename markierten Stelle beginnt. Fehlt die Lochung, so wird die nächste Instruktion ausgeführt. Da die Kanalbedingungen nicht vor dem Geräteende signalisiert werden, sollte die Druckeinstellung erst unmittelbar vor dem nächsten Druck-, Zeilentransport- oder Vorschubbefehl getestet werden. Auf eine Prüfung nach den erwähnten Befehlen sollte verzichtet werden, da eine im Gang befindliche Operation erst vervollständigt sein muß, bevor der Test gemacht wird.

Wenn der letzte Operand weggelassen wird, erfolgt auf Veranlassung des IOCS automatisch ein Vorschub bis zu einer Lochung im Kanal 1.

EA-Gerät	code	n	m	*Bedeutung*
Magnetband-gerät	REW	-	-	Zurückspulen des Bandes
	RUN	-	-	Zurückspulen und Entladen des Bandes
	ERG	-	-	Löschen Kluft
	WTM	-	-	Schreiben Bandmarke
	BSR	-	-	Zurücksetzen bis zur nächsten Kluft
	BSF	-	-	Zurücksetzen bis zur nächsten Bandmarke
	FSR	-	-	Vorwärtssetzen bis zur nächsten Kluft
	FSR	-	-	Vorwärtssetzen bis zur nächsten Bandmarke
	BSL	-	-	Zurücksetzen um einen logischen Satz
	FSL	-	-	Vorwärtssetzen über einen logischen Satz hinweg
Zeilendrucker	SP	a	b	Zeilentransport um a Zeilen vor bzw. b Zeilen nach dem Drucken
	SK	a	b	Sofortvorschub bis zu einer Lochung im Kanal a bzw. Vorschub nach dem Drucken bis zu einer Lochung im Kanal b
Zeilendrucker mit Universaltypensatz	UCS	ON	-	Durchführung der Datenprüfung
	UCS	OFF	-	Verhinderung der Datenprüfung
Zeilendrucker IBM 3211	FOLD	-	-	Drucken von Großbuchstaben bei Bytes mit Kleinbuchstaben
	UNFOLD	-	-	Drucken von Kleinbuchstaben bei Bytes mit Kleinbuchstaben
IBM 2540	PS	c	-	Auswahl des Ablagefaches c (c = 1, 2, 3)
IBM 3504, 3505, 3525				Auswahl des Ablagefaches c (c = 1, 2, 3; bei c = 3 wird Ablagefach 2 gewählt)
IBM 2560	SS	c	-	Auswahl des Ablagefaches c c = 1, 2, 3, 4, 5)
IBM 5425				Auswahl des Ablagefaches c (c = 1, 2, 3, 4)
IBM 2520, 1442				Auswahl des Ablagefaches c (c = 1 oder c = 2)
IBM 1442	E	-	-	Auslauf einer Karte
Alle SEDZ	SEEK	-	-	Suchen nach der Adresse (Einstellen auf eine Adresse)

Abb. 21.2. Übersicht über die Operanden des Makrostatements CNTRL und ihre Bedeutung

In der durch PRTOV angesprungenen Routine darf mit Ausnahme eines weiteren PRTOV-Makros jede beliebige andere Instruktion und jedes beliebige andere Makro gegeben werden. Am Ende dieser Routine muß der Programmierer eine Rückkehr zum IOCS vorsehen, indem er zu der Adresse verzweigt, die das IOCS ins Register 14 gestellt hat, bevor es die Benutzerroutine angesprungen hat. — Eine gute Programmierpraxis vermeidet die Verwendung des Makros PRTOV, da es die Leistungsfähigkeit eines Programmes vermindert. Stattdessen sollte ins Programm ein Zähler eingeführt werden, mit dem die einzelnen Zeilen einer Seite verfolgt werden.

Einige andere Makros können nur auf Magnetbandgeräte und auf Speichergeräte mit direkter Zugriffsmöglichkeit bezogen werden. Wenn nichtstandardmäßige Kennsätze oder Benutzerkennsätze zu verarbeiten sind, so muß der Benutzer nach dem Prüfen oder Schreiben eines Kennsatzes eine Rückkehr zum IOCS veranlassen, indem er das Makrostatement

```
marke  LBRET n
```

verwendet. Unter n ist 1 zu codieren, wenn keine weiteren Kennsätze geprüft oder geschrieben werden sollen; es ist jedoch 2 niederzuschreiben, wenn noch weitere Kennsätze geprüft oder aufgezeichnet werden sollen.

Das Makro

```
marke  RELSE dateiname
```

wird im Zusammenhang mit geblockten Sätzen verwendet. Es bewirkt, daß die in einem Block verbleibenden Sätze übergangen werden. Die Verarbeitung wird mit dem ersten Satz des nächsten Blockes fortgesetzt.

Normalerweise zeichnen die Routinen des logischen IOCS erst dann einen Satzblock auf, wenn der Block voll ist oder der nächste Satz nicht mehr in den Block hineinpaßt. Teilweise gefüllte Blöcke können durch das Makro

```
marke  TRUNC dateiname
```

geschrieben werden. Wenn Satzblöcke variabler Länge aufgezeichnet werden sollen, muß das Makro TRUNC verwendet werden, da durch PUT-Makros die Blöcke im allgemeinen nicht exakt aufgefüllt werden.

Bei Magnetbandgeräten kann das Ende eines Datenträgers durch das Makro

```
marke  FEOV  dateiname
```

erzwungen werden. Bei einer Eingabedatei wird die Verarbeitung mit dem nächsten Datenträger fortgesetzt. Bei einer Ausgabedatei wird der letzte Satzblock aufgezeichnet, diesem folgen die Bandmarke und die Nachsätze. FEOV bewirkt zum Schluß automatisch ein Rückspulen des Magnetbandes und geht anschließend zum nächsten Datenträger über. Die FEOV-Routine ist identisch mit der Routine, die bei einer Ausgabedatei dann angesprungen wird, wenn das physikalische Ende des Magnetbandes entdeckt wird, und bei einer Eingabedatei dann, wenn eine Bandmarke festgestellt wird.

Bei einem Plattenspeichergerät kann das Auftreten der Endbedingung für einen Bereich mittels des Makros

```
marke  FEOVD dateiname
```

erzwungen werden.

Bekommt eine Fehlerbehandlungsroutine die Programmablaufsteuerung übertragen, um einen Band- oder Plattenfehler zu untersuchen, so muß die Steuerung an das IOCS mittels des Makros

```
marke   ERET   operand
```

zurückgegeben werden.

Wird dabei für operand SKIP niedergeschrieben, so wird der fehlerhafte Block übergangen und die Verarbeitung mit dem nächsten Block fortgesetzt. Die Codierung von IGNORE hat zur Folge, daß der Fehler unbeachtet bleibt. Bei Plattenspeichern wird SKIP als IGNORE angesehen. Bei der Niederschrift von RETRY wird dafür gesorgt, daß die EA-Operation erneut versucht wird. RETRY ist jedoch ungültig, wenn bei Plattenspeichern Sätze falscher Länge anstehen; bei Magnetbändern sogar bei jedem Satz. Fehlt die Codierung eines Operanden, so wird standardmäßig SKIP angenommen.

Während der Ausführung der EA-Routinen werden die Inhalte der Register 0, 1, 14 und 15 geändert. Somit müssen die Informationen, die in diesen Registern stehen und später wieder gebraucht werden, gerettet werden, bevor das Makro gegeben wird. Zusätzlich wird das Register 13 beim SAVE-Makro (siehe Abschnitt 21.19.) verwendet. Infolgedessen kann der Programmierer unbeschränkt nur über die Register 2 bis 12 verfügen.

Bei den in diesem Abschnitt behandelten Makros kann der Ausdruck dateiname durch (r) ersetzt werden, wobei r ein Ausdruck ist, dessen Wert zwischen 2 und 12 einschließlich liegt. Zusätzlich kann auch (1) benutzt werden, jedoch nicht bei den beiden Makros OPEN und CLOSE. Wenn (r) oder (1) codiert wird, so muß sich die Adresse des Dateinamens (Marke bei der DTF-Eintragung) im angegebenen Register r oder 1 befinden.

Bei den Makros GET und PUT kann auch arbeitsgebietname durch eine Registerangabe (r) oder (Ø) ersetzt werden, sofern zwar die Adresse des Arbeitsgebietes in das spezifizierte Register geladen wurde.

Beim Makro PRTOV kann auch routinename zu (r) oder (Ø) geschrieben werden, wobei auch hier darauf zu achten ist, daß die Adresse der Routine im aufgeführten Register r oder Ø steht.

Als Beispiele wollen wir die in Abb. 21.3. dargestellten neun Statements ansehen, die alle gültig sind.

```
        GET   CF,DR
        GET   CF,(2)
        GET   CF,(Ø)
        GET   (5),DR
MARI    GET   (5),(2)
        GET   (5),(Ø)
AN      GET   (1),DR
NE      GET   (1),(2)
        GET   (1),(Ø)
```

Abb. 21.3. Beispiele für gültige GET-Statements

Zur Abb. 21.3. erscheinen uns einige Kommentare als angemessen.

Selbstverständlich muß zuvor die Adresse von CF in das Register 5 bzw. in das Register 1 geladen werden. In den Fällen, bei denen die entsprechende Angabe aufgeführt ist, muß natürlich auch die Adresse von DR entweder ins Register 0 oder ins Register 2

geladen werden. Die echten Instruktionen, die vom Assemblierer des DOS aus der ersten Makroinstruktion generiert werden, sind die folgenden:

```
L       1,=A(CF)
L       Ø,=A(DR)
L       15,16(1)
BAL     14,8(15)
```

Durch die letzten beiden Instruktionen wird die Standardverbindung zur logischen IOCS-Routine hergestellt. Die ersten beiden Instruktionen werden durch die Instruktionen

```
LR      1,5
LR      Ø,2
```

ersetzt, wenn als Operanden die Registerangaben (r) erscheinen (nur bei den Registerangaben 5 bzw. 2). Wenn (1) bzw. (Ø) codiert ist, dann verschwinden diese beiden Instruktionen gänzlich; auf diese Weise wird etwas Hauptspeicherplatz eingespart.

Zum Schluß dieses Abschnittes wollen wir darauf hinweisen, daß bei allen bis jetzt behandelten Makrostatements die Angabe von marke freigestellt ist.

21.4. Die Dateidefinition DTFCD

Das Makro DTFCD[1)] wird gewöhnlich zur Definition von Dateien benutzt, die auf Lochkarten vorliegen. In der Abb. 21.4. sind die verschiedenen Operanden tabelliert, die bei diesem Statement erlaubt sind. Es werden drei Dateiarten (TYPEFLE) unterstützt: Eingabedateien, Ausgabedateien und kombinierte Dateien. Unter einer kombinierten Datei wollen wir eine Datei verstehen, bei der Karten gelesen werden und anschließend in die gleichen Karten Informationen gestanzt werden (Stanzen in gleiche Karte).

Bei der Abb. 21.4. wie auch bei den folgenden Abbildungen gleicher Art sind die Standardangaben unterstrichen, d.h. diese Angaben werden benutzt, wenn im Statement der betreffende Operand fehlt. Im übrigen dürfen die Operanden in einer beliebigen Reihenfolge niedergeschrieben werden.

Die Eingabe ist beschränkt auf ungeblockte Sätze fester Länge. Ausgabesätze können daneben auch noch eine undefinierte Länge aufweisen oder zur Satzart „ungeblockt mit variabler Länge'' gehören.

EA-Puffer sind vom Programmierer bereitzustellen. Durch die Eintragung

```
BLKSIZE=n
```

wird dabei die Länge des EA-Gebietes festgehalten, die vom Programm der IOCS-Routine zur Verfügung gestellt wird. Wenn die Sätze unterschiedliche Länge aufweisen, so muß das EA-Gebiet groß genug sein, um auch den längsten Satz aufnehmen zu können.

Die Adresse des EA-Gebietes wird durch die Eintragung

```
IOAREA1=name
```

erfaßt. Wenn zwei EA-Gebiete benutzt werden sollen, so muß die Adresse des zweiten Gebietes in der Eintragung

```
IOAREA2=name
```

---

1) Abkürzung von ,,Define the File for Cards" (,,Definieren einer Kartendatei")

Operand	*Bedeutung*	*Bemerkungen*
ASOCFLE=name	Verknüpfte Datei	Name der verknüpften Datei (DTFCD oder DTFPR)
BLKSIZE=n	Länge des EA-Gebietes	Bei Weglassung dieser Angabe wird standardmäßig 80 oder 96 für n angenommen; beim binären Lesen 160
CONTROL=YES	Das Makro CNTRL wird v verwendet	Bei Angabe dieses Operanden muß der Operand CTLCHR weggelassen werden
CRDERR=RETRY	Wiederaufnahme nach Stanzfehlern (Neustanzen fehlerhafter Karten)	Gilt nur für IBM 2520 und IBM 2540
CTLCHR={YES / ASA}	Das erste Zeichen eines Satzes ist ein Gerätesteuerzeichen	YES → Maschinensteuerzeichen ASA → Steuerzeichen nach ASA
DEVADDR=SYSnnn	Festlegung des logischen Gerätenamens	Erforderlich für alle Dateien
DEVICE=n	Gerätetyp n = 1442, 2501, 2520, 2540, 2560 P, 2596, 3504, 3525, 5425 P, 5425 S	Standardmäßig wird bei Fehlen dieser Angabe 2540 unterstellt
EOFADDR=name	Verzweigungsadresse bei Feststellung des Dateiendes	Erforderlich bei Eingabedateien und bei kombinierten Dateien. Unter name ist ein Adreßausdruck zu verstehen
ERROPT={IGNORE / SKIP / name}	Behandlung von EA-Fehlern (name Verzweigungsadresse)	Nur bei den Geräten IBM 2560, 3504, 3505, 3525 und 5425 angebbar; nicht für kombinierte Dateien
FUNC=x	x = R, P, W, I, R P, RW, RPW, PW	Nur bei den Geräten 2560, 3535, 5425 angebbar
IOAREA1=name	EA-Gebiet mit der Adresse name	Erforderliche Angabe für jede Datei
IOAREA2=name	EA-Gebiet mit der Adresse name	Festlegung eines 2. EA-Gebietes; kombinierte Dateien
IOREG=(r)	Adresse des EA-Gebietes im Register r (r zwischen 2 und 12 einschließlich)	Voraussetzung: Zwei EA-Geräte und kein Arbeitsgebiet

Abb. 21.4. Mögliche Operanden bei der Dateidefinition DTFCD (Teil 1)

Operand	*Bedeutung*	*Bemerkungen*
MODE=x	Arbeitsmodus a) x = E oder C bei IBM 2560 b) x = E, C, R, ER oder CR bei IBM 3525 c) x = E, C, O, R, EO, ER, CO oder CR bei IBM 3505 und IBM 3505	Erforderliche Angabe, wenn ein anderer Modus als E vorgesehen ist
OUBLKSZ=n	Länge n des zweiten Ausgabegebietes bei kombinierten Dateien	Fehlt dieser Operand, so wird standardmäßig der bei BLKSIZE angegebene Wert angenommen
RECFORM={FIXUNB / UNDEF / VARUNB}	Satzart (Satzform)	VARUNB und UNDEF dürfen nur bei Ausgabedateien angegeben werden; Standardangabe ist FIXUNB
RECSIZE=(r)	Register r (2 bis 12 einschließlich) enthält die Satzlänge	Bei Dateien mit der Satzart UNDEF erforderliche Angabe
SSELECT=n	Fachauswahl (n ist gleich der Ablagefachnummer)	Wenn nicht angegeben, werden NR oder NP (Normalablagefächer beim Leser bzw. beim Stanzer) angenommen
TYPEFLE={INPUT / OUTPUT / CMBND}	Festlegung der Dateiverwendung (Eingabe, Ausgabe, kombinierte Eingabe/Ausgabe)	CMBND nur bei den Gerätetypangaben 1442 N1, 2520 B1 oder 2540 P möglich
WORKA=YES	Arbeitsgebiet bei GET- bzw. PUT-Makrostatements	a) Nicht zusammen mit IOREG angebbar b) Nicht bei Ausgabe erlaubt, wenn ERROPT = IGNORE angegeben ist.

Abb. 21.4. Mögliche Operanden bei der Dateidefinition DTFCD (Teil 2)

erscheinen. Wenn nur ein EA-Gebiet verwendet wird, dann wird der Operand IOAREA2 nicht codiert.

Wenn ein Arbeitsgebiet benutzt wird, so muß

```
WORKA=YES
```

geschrieben werden. Die echte Adresse des Arbeitsgebietes wird in diesem Makro nicht angegeben, da bei einer Datei in den verschiedenen GET- bzw. PUT-Statements auch verschiedene Arbeitsgebiete spezifiziert werden können.

Wenn zwei EA-Gebiete verwendet werden, wird die Adresse des augenblicklich benutzten Gebietes in das Register (2 bis einschließlich 12) gestellt, das durch

IOREG=(r)

festgelegt wurde. Somit bedeutet z. B.

IOREG=(4) ,

daß bei der Ausführung eines GET-Statements die Adresse des nächsten logischen Satzes im Register 4 steht. In gleicher Weise würde ein OPEN-Statement das Register 4 auf die Adresse setzen, auf der das Programm mit dem Aufbau einer Ausgabedatei beginnen kann, deren Sätze durch PUT-Statements gestanzt werden.

Bei Daten mit Sätzen undefinierter Länge wird durch

RECSIZE=(r)

das Register angegeben, das die Länge des Ausgabesatzes enthält, wenn das PUT-Statement zur Ausführung gelangt. Unter r darf irgendeines der Register 2 bis einschließlich 12 angegeben werden.

Die Eintragung CTLCHR gehört nur zu Ausgabedateien. Sie informiert das System, daß das erste Byte jeden Satzes ein Steuerzeichen enthält (ASA-Steuerzeichen oder Maschinensteuerzeichen). In der Regel wird auf diese Weise die Fachauswahl bewirkt.

Wird

CONTROL=YES

angegeben, so kann das Makrostatement CNTRL benutzt werden. CNTRL und CTLCHR können nicht gleichzeitig auf eine Datei bezogen werden.

Durch ERROPT wird die Programmstelle festgelegt, zu der die Ablaufsteuerung übertragen wird, wenn ein Eingabe- oder ein Ausgabefehler entdeckt wird. Alternativ kann IGNORE (erlaubt die Verarbeitung so, als ob nichts falsch gewesen wäre) oder SKIP (Übergehen der fehlerhaften Karte) festgelegt werden.

Bei MODE können folgende Kennbuchstaben aufgeführt werden:

E	EBCDI-Code
C	Dualkartenverarbeitung
O	Optisches Markierungslesen
R	Lesen mit Spaltenunterdrückung

Falls C nicht angegeben ist, wird stets E angenommen. Bei Ausgabedateien kann sowieso nur E oder C aufgeführt werden. Bei den logischen Dateien SYSIPT, SYSRDR und SYSPCH ist nur der Buchstabe E akzeptabel. Es sei am Rande vermerkt, daß beim binären Lesen die Zeichenkombinationen /* und /& nicht als solche identifiziert werden können. Beim binären Lesen ist zur Beendigung desselben unbedingt erforderlich, daß das physikalische Ende einer Datei festgestellt wird.

Bei einer kombinierten Datei wird das unter IOAREA1 angegebene Gebiet als Eingabepuffer und das unter IOAREA2 angegebene als Ausgabepuffer benutzt. Fehlt die Angabe des Operanden IOAREA2, so wird das bei IOAREA1 genannte Gebiet für beide Zwecke eingesetzt.

Durch FUNC wird Lesen (R), Stanzen (P) und Drucken (W) in einer beliebigen Kombination für diejenigen Geräte festgelegt, die zu einer solchen Operation in der Lage sind. Die Angabe von I verursacht das Stanzen und Drucken der gleichen Information; nur ein EA-Puffer ist bei Nennung dieses Kennbuchstabens erlaubt. Wird bei FUNC mehr als ein Kennbuchstabe spezifiziert, also beispielsweise RP, RW, PW oder RPW, dann müssen verknüpfte Dateien definiert werden, eine Datei fürs Lesen, eine fürs Stanzen

und/oder eine fürs Drucken. In diesem Fall muß jedes DTFCD-Makrostatement im Operanden ASOCFLE den Namen eines anderen DTFCD-Makrostatements enthalten. Das DTFCD-Makro für die Lesedatei nennt also den Namen des DTFCD-Makros für die Stanzdatei, falls eine solche vorhanden ist, anderenfalls den Namen der unter DTFPR definierten Druckdatei. Das DTFCD-Makro für die Stanzdatei verweist auf den Namen des DTFPR-Makros der Druckdatei, falls es eine solche gibt, anderenfalls auf den Namen der mit DTFCD definierten Lesedatei. Das Makro DTFPR für die Druckdatei wiederum zeigt auf den Namen des DTFCD-Makros für die Lesedatei, falls es eine solche gibt, anderenfalls auf den Namen des DTFCD-Makros für die Stanzdatei.

Wir wollen noch einmal wiederholen, was wir bereits zu Anfang dieses Abschnittes sagten. Die Operanden der Dateidefinition DTFCD können in einer beliebigen Reihenfolge geschrieben werden; es können auch beliebig viele Fortsetzungszeilen benutzt werden. Man wird bemerkt haben, daß in der Abb. 21.4. einige Angaben unterstrichen sind. Die Unterstreichungen bedeuten, daß bei Weglassung des entsprechenden Operanden der unterstrichene Wert angenommen wird. Somit bedeutet z. B. die Auslassung von

```
TYPEFLE
```

daß standardmäßig TYPEFLE=INPUT angenommen wird.

Die Dateidefinition DTFCD in der Abb. 21.5. dient z. B. zum Einlesen eines Kartenpaketes; sie verwendet dabei EBER1 als Puffergebiet.

```
CD      DTFCD DEVICE=25Ø1,                    1
              DEVADDR=SYSIPT,                 2
              EOFADDR=BR2,                    3
              BLKSIZE=8Ø,                     4
              IOAREA1=EBER1,                  5
              WORKA=YES
```

Abb. 21.5. Beispiel für die Dateidefinition DTFCD

In den GET-Makros ist Bezug auf ein Arbeitsgebiet zu nehmen. Wenn die Dateiendebedingung festgestellt wird, verzweigt das IOCS zu einer Routine, die auf dem Speicherplatz mit der symbolischen Adresse BR2 beginnt. Im Programm sind die folgenden Makros zu verwenden:

```
OPEN  CD
GET   CD,ARBNAME
CLOSE CD
```

Unter ARBNAME ist hierbei der (willkürlich gewählte) Name eines Arbeitsgebietes zu verstehen.

21.5. Die Dateidefinition DTFPR

Druckdateien werden mit Hilfe des Makros DTFPR[1)] definiert. Die möglichen Operanden sind in der Abb. 21.6. aufgeführt. Die Eintragungen entsprechen im wesentlichen denen von DTFCD.

1) Abkürzung von „Define the File for Printer" („Definieren einer Druckerdatei")

Operand	*Bedeutung*	*Bemerkungen*
ASCOFLE=name	verknüpfte Datei	Name der verknüpften Datei
BLKSIZE=n	Länge des EA-Gebietes	Standardlängen sind: a) 121 für Zeilendrucker b) 64 für IBM 2560 und IBM 3525 c) 96 für IBM 5203 oder IBM 5425
CONTROL=YES	Das Makro CNTRL wird verwendet	Bei Angabe dieses Operanden muß der Operand CTLCHR weggelassen werden
CTLCHR={YES / ASA}	Das erste Zeichen eines Satzes ist ein Gerätesteuerzeichen	YES → Maschinensteuerzeichen ASA → Steuerzeichen nach ASA
DEVADDR=SYSnnn	Festlegung des logischen Gerätenamens	Erforderlich für alle Dateien
DEVICE=n	Gerätetyp n = 1403, 1443, 2560P, 2560S, 3203, 3211, 3525, 5203, 5425P, 5425S	Standardmäßig wird bei Fehlen dieses Operanden DEVICE= 1403 unterstellt
ERROPT={RETRY / IGNORE / name}	Behandlung von EA-Fehlern (name Verzweigungsadresse)	a) IGNORE nur für IBM 3525 b) RETRY oder name nur für IBM 3211 möglich c) Für alle anderen Geräte nicht möglich
FUNC=x	x = W, RW, PRW, PW, WT, RWT, PPWT, PWT	Nur bei den Geräten IBM 2560, 5425 und 3525 angebbar
IOAREA1=name	EA-Gebiet mit der Adresse name	Erforderliche Angabe für jede Datei
IOAREA2=name	EA-Gebiet mit der Adresse name	Festlegung eines 2. EA-Gebietes
IOREG=(r)	Adresse des EA-Gebietes im Register r (r zwischen 2 und 12 einschließlich)	Voraussetzung: Zwei EA-Gebiete und kein Arbeitsgebiet
PRINTOV=YES	Das Makro PRTOV wird verwendet	Nicht für IBM 2560 und IBM 5425 angebbar
RECFORM={FIXUNB / VARUNB / UNDEF}	Satzart (Satzform)	Diese Angabe ist für alle Dateien erforderlich, deren Satzform nicht FTXUNB ist
RECSIZE=(r)	Register r (2 bis einschließlich 12) enthält die Satzlänge	Bei Dateien mit der Satzart UNDEF erforderliche Angabe
UCS={OFF / ON}	Verarbeitung bei Datenfehlern	OFF: Sperrung der Datenprüfung ON: Durchführung der Datenprüfung
WORKA=YES	Arbeitsgebiet bei PUT-Makrostatements	Nicht zusammen mit IOREG angebbar

Abb. 21.6. Mögliche Operanden bei der Dateidefinition DTFPR

Die Benutzung des Operanden CTLCHR ist anzustreben, dagegen sollte die der beiden Operanden PRINTOV und CONTROL eingeschränkt werden. Die beste Programmierpraxis wird dadurch erzielt, daß man nur Vorschübe zum Kanal 1 vorsieht und im übrigen die Zeilen einer Seite fortlaufend zählt. Es ist nicht empfehlenswert, das Ende einer Seite mittels des Makros PRTOV zu ermitteln.

Bei verknüpften Dateien ist T bis einer IBM 3525 zu spezifizieren, wenn diese mit einem Zweizeilendrucker ausgerüstet ist. Wird bei einer IBM 3525 T nicht angegeben, so wird Mehrzeilendrucken angenommen. T bleibt unbeachtet, wenn CONTROL oder CTLCHR angegeben sind.

Von Wichtigkeit ist die Bemerkung, daß bei Aufführung von

```
CONTROL=YES
```

die Angabe von CTLCHR weggelassen werden muß und umgekehrt, d.h. entweder kann das Makro CNTRL verwendet werden oder ein Steuerzeichen, aber niemals beide. Meist wird man sich für die Verwendung des Steuerzeichens entscheiden. Tatsächlich wird bei Druckdateien fast immer ein Steuerzeichen benutzt, bei anderen Dateien dagegen höchst selten.

21.6. Die Dateidefinition DTFCN

Die Dateidefinitionen DTFCN[1)] werden benutzt, um Eingaben und Ausgaben festzulegen, die für das Systemsteuerpult gelten. Zusätzlich zu den bereits besprochenen Makrostatements GET und PUT (Abschnitt 21.3.) gibt es zusätzlich für diese EA ein weiteres Makro. Es lautet allgemein:

```
marke   PUTR   dateiname
marke   PUTR   dateiname,arbeitsgebiet1,arbeitsgebiet2
```

Die Verwendung von marke ist auch bei diesem Statement freigestellt.

Bei einer Konsole des Typs IBM 3210 oder IBM 3215 entspricht das Makro PUTR einer Ausgabe vom Arbeitsgebiet 1, der sich eine Eingabe ins Arbeitsgebiet 2 anschließt. Sind keine Arbeitsgebiete spezifiziert, so wird der erste Teil des bei IOAREA1 genannten Gebietes für die Ausgabe und der Rest dieses Gebietes für die Eingabe benutzt.

Der maximale Wert, der bei BLKSIZE aufgeführt werden kann, ist 256 Bytes.

Bei einer mit Bildschirm ausgerüsteten Konsole wirkt das PUTR-Makro genau so wie es bei den Gerätetypen IBM 3210 und IBM 3215 der Fall ist. Zusätzlich kennzeichnet es die Ausgabenachricht, daß sie erst dann zerstört werden darf, wenn die Antwort des Maschinenbedieners (Operateurs) vorliegt.

Unter arbeitsgebiet 1 kann wie üblich der Name eines Gebietes genannt werden; es ist jedoch auch eine Registerangabe in der Form (Ø) oder (r) möglich, wobei r ein Ausdruck sein muß, dessen Wert zwischen 3 und 12 einschließlich liegt. Dasselbe gilt für den Operanden arbeitsgebiet 2. Auch für dateiname ist eine Registerspezifikation möglich, entweder zu (1) oder zu (r), wobei r Werte zwischen 3 und 12 einschließlich annehmen muß.

Beim Mehrprogrammbetrieb geht den für SYSLOG bestimmten Nachrichten ein Präfix voraus (BG, F1, F2 usw.). Der Präfix fehlt bei den für SYSnnn mit nnn = 000, · · ·, 221 bestimmten Nachrichten.

1) Abkürzung von „Define the File for Console" (Definieren einer Konsoldatei")

Bei Dateien mit dem INPUT-Vermerk können sowohl GET- als auch PUT-Statements verwendet werden; Dateien mit dem Vermerk OUTPUT gestatten allein die Verwendung von PUT-Statements. Sowohl GET- und PUT- als auch PUTR-Statements dürfen für Dateien mit der Eintragung CMBND gegeben werden.

Sollte dem Operateur während der Eingabe ein Fehler unterlaufen, so kann er die Eingabe löschen und von neuem beginnen. Geht er in dieser Weise vor, so wird das Eingabegebiet nicht zuvor auf Leerzeichen oder duale Nullen gebracht. Aus diesem Grund ist es mitunter leichter, Dateien im Programm zu handhaben, die mit der Satzform UNDEF erklärt sind.

Die möglichen Operanden für das Makrostatement DTFCN sind in der Abb. 21.7. zusammengestellt.

Operand	*Bedeutung*	*Bemerkungen*
BLKSIZE=n	Länge des EA-Gebietes (bei PUTR Länge des Ausgabeteiles des EA-Gerätes)	Erforderlich nur, wenn eine andere Länge als 80 Bytes vorgesehen ist
DEVADDR={SYSLOG / SYSnnn}	Festlegung des logischen Gerätenamens	Bei Verwendung von PUTR ist die Angabe von SYSLOG notwendig
INPSIZE=n	Länge des Eingabeteiles des EA-Gebietes bei PUTR	Erforderlich bei Verwendung von PUTR
IOAREA1=name	EA-Gebiet mit der Adresse name	Erforderliche Angabe für jede Datei
RECFORM={FIXUNB / UNDEF}	Satzart (Satzform)	Diese Angabe ist für alle Dateien erforderlich, deren Satzform nicht FIXUNB ist
RECSIZE=(r)	Register I (2 bis einschließlich 12) enthält die Satzlänge	Bei Dateien mit der Satzart UNDEF erforderliche Angabe
TYPEFLE={INPUT / OUTPUT / CMBND}	a) INPUT verarbeitet sowohl Eingabe- als auch Ausgabeoperationen b) OUTPUT verarbeitet nur Ausgabeoperationen	a) PUTR erfordert die Angabe von CMBND b) Diese Angabe ist nicht erforderlich, wenn INPUT gelten soll
WORKA=YES	Arbeitsgebiet bei den Makrostatements GET, PUT und PUTR	- -

Abb. 21.7. Mögliche Operanden bei der Dateidefinition DTFCN

21.7. Die Dateidefinition DTFMT

In der Abb. 21.8. sind die Operanden zusammengestellt, die bei der Dateidefinition DTFMT[1]) möglich sind. Drei verschiedene Dateiverwendungen können vorgesehen werden:

a) Eingabedateien
b) Ausgabedateien
c) Arbeitsdateien

Im Augenblick wollen wir uns um Arbeitsdateien nicht kümmern; wir werden das im Abschnitt 21.10. nachholen.

Operand	*Bedeutung*	*Bemerkungen*	
BLKSIZE=n	Länge des EA-Gerätes	Für jede Datei erforderlich	*
DEVADDR=SYSnnn	Festlegung des logischen Gerätenamens	Für jede Datei erforderlich	*
EOFADDR=name	Verzweigungsadresse bei Dateiende	Für Eingabedateien und für Arbeitsdateien erforderlich	*
ERREXT=YES	Benutzung des Makros ERET	--	*
ERROPT={IGNORE / SKIP / name}	Behandlung von EA-Fehlern (name Verzweigungsadresse)	IGNORE bewirkt, daß vom IOCS die Fehler vernachlässigt werden; SKIP veranlaßt, daß das IOCS den Satz übergeht; unter name ist die Adresse anzugeben, zu der das IOCS bei Fehlern verzweigen soll	*
FILABL={STD / NSTD / NO}	Kennsatzart (standardisiert, nichtstandardisiert)	a) Für jede Datei erforderlich, die Kennsätze aufweist b) NSTD erfordert die Angabe des Operanden LABADR c) Arbeitsdateien erfordern die Angabe von NO oder von STD	*
HDRINFO=YES	Ausgabe von Informationen über den Vorsatz	Voraussetzung: FILABL = STD	

1) Abkürzung von „Define the File for Magnetic Tape" („Definieren einer Magnetbanddatei")

Operand	*Bedeutung*	*Bemerkungen*	
IOAREA1=name	Adresse des EA-Gebietes	Für jede Datei erforderlich; unter name kann ein Adreßausdruck angegeben werden	
IOAREA2=name	Adresse des zweiten EA-Gebietes	Festlegung eines zweiten EA-Gebietes	
IOREG=(r)	Bei zwei EA-Gebieten: Register zur Steuerung der EA	r = 2 bis r = 12; Anwendung auch bei geblockten Sätzen, wenn diese im EA-Gebiet verarbeitet werden sollen	
LABADDR=name	Verarbeitung von Benutzerkennsätzen oder von nichtstandardisierten Kennsätzen (Vorsätzen)	Unter name ist die Adresse der Benutzerroutine zur Kennsatzverarbeitung aufzuführen	
NOTEPNT={YES / POINTS}	Verwendung verschiedener Makros bei Arbeitsdateien	a) bei YES: NOTE, POINTR; außerdem ist auch die Verwendung von POINTS möglich b) bei POINTS: nur Makro POINTS	*
READ={FORWARD / BACK}	Lesemethode: vorwärts bzw. rückwärts	Fehlt diese Angabe, so gilt FORWARD	*
RECFORM={FIXUNB / FIXBLK / VARUNB / VARBLK / SPNUNB / SPNBLK / UNDEF}	Satzart (Satzform)	a) Erforderlich nur, wenn ein anderes Satzformat als FIXUNB vorliegt b) Bei Arbeitsdateien sind nur FIXUNB und UNDEF erlaubt	*
RECSIZE={n / (r)}	Satzlänge	Durch r kann eines der Register 2 bis einschließlich 12 angegeben werden	

Abb. 21.8. Mögliche Operanden bei der Dateidefinition DTFMT (1. Teil)

Operand	*Bedeutung*	*Bemerkungen*	
REWIND={UNLOAD / NORWD}	Angaben für das Zurückspulen von Magnetbändern a) UNLOAD → Zurückspulen b) NORWD → kein Zurückspulen	Standardmäßig erfolgt das Zurückspulen	*
TPMARK=NO	Schreiben von Bandmarken am Ladepunkt	a) Gilt nicht für Arbeitsdateien b) Das Schreiben von Bandmarken unterbleibt bei Angabe von FILABL=NSTD und bei Angabe von FILABL=NO	
TYPEFLE={INPUT / OUTPUT / WORK}	Festlegung der Dateiverwendung (Eingabe-, Ausgabe-, Arbeitsdatei)	Dieser Operand kann bei Eingabedateien entfallen	*
VARBLD=(r)	Verbleibende Anzahl von Bytes im augenblicklich benutzten Ausgabegebiet bei Dateien mit Sätzen variabler Länge	a) Durch r kann eines der Register 2 bis 12 angegeben werden b) Voraussetzungen sind TYPEFLE=OUTPUT und kein Arbeitsgebiet	
WLRERR=name	Verarbeitung von Eingabesätzen mit falscher Länge (name ist dabei der Name der Eingangsstelle in die entsprechende Routine)	Bei Weglassung dieser Angabe werden derartige Sätze so behandelt, wie es bei ERROPT spezifiziert wurde; wenn auch ERROPT weggelassen ist, wird die Bedingung „falsche Satzlänge" ignoriert	
WORKA=YES	Verwendung eines Arbeitsgebietes (bei GET oder PUT)	a) Für Arbeitsdateien nicht erlaubt b) Nicht zusammen mit IOREG angebbar	

Anmerkung Die mit einem Sternzeichen versehenen Operanden sind auch bei Arbeitsdateien erlaubt

Abb. 21.8. Mögliche Operanden bei der Dateidefinition DTFMT (2. Teil)

Die meisten Eintragungen, die beim Makrostatement DTFMT möglich sind, entsprechen denen, die wir bei den Makrostatements DTFCD und DTFPR kennengelernt haben. Einen Gesamtüberblick über die Operanden zeigt uns die Abb. 21.8. Einzelne Operanden bedürfen, da sie uns noch nicht bekannt sind, der Erläuterung.

Die Verwendung des Operanden HDRINFO bewirkt, daß der Inhalt des Kennsatzes mit der Kennung HDR1 über SYSLOG immer dann ausgegeben wird, wenn die betreffende Datei eröffnet wird. Dieser Operand wird gewöhnlich weggelassen.

Durch den Operanden LABADDR wird die Eingangsadresse von derjenigen Routine genannt, die zur Verarbeitung von nichtstandardisierten oder von Benutzervorsätzen herangezogen werden soll. Eine solche Benutzerroutine muß mit dem Makro LBRET abgeschlossen sein. Ausgabekennsätze werden im regulären EA-Gebiet der betreffenden Datei zusammengestellt.

Das Betriebssystem DOS ist so gestaltet, daß es üblicherweise auf dem Magnetband als ersten Datensatz eine Bandmarke aufzeichnet, wenn keine Standardkennsätze benutzt werden. Das Schreiben dieser Bandmarke kann durch Angabe des Operanden

```
TPMARK=NO
```

unterdrückt werden. Im Betriebssystem OS wird diese Bandmarke am Ladepunkt üblicherweise nicht aufgezeichnet.

Der Operand

```
RECSIZE=(r)
```

wird gewöhnlich bei Dateien verwendet, die aus Sätzen nicht definierter Länge bestehen; durch sie wird das Register angegeben, das die Satzlänge enthält. Bei geblockten Sätzen fester Länge gibt die Eintragung

```
RECSIZE=n
```

(ohne Klammern geschrieben) die Länge eines jeden Satzes an. Zusammen mit der Eintragung BLKSIZE wird dadurch das IOCS in die Lage versetzt, den Blockungsfaktor zu ermitteln.

Im Betriebssystem DOS läuft bezüglich der Magnetbandbehandlung im Normalfall das folgende Verfahren ab: Das Magnetband wird bis zum Ladepunkt zurückgespult, wenn eine der Operationsfolgen abläuft, die durch die Makros OPEN und CLOSE veranlaßt, werden. Der Operand

```
REWIND=NORWD
```

unterdrückt jedoch das Zurückspulen. Die Form

```
REWIND=UNLOAD
```

hat zur Folge, daß beim Eröffnungsprozeß das Zurückspulen des Bandes durchgeführt wird, beim Schließungsprozeß wird daran jedoch noch ein Entladen gekoppelt.

Der Operand

```
ERREXT=YES
```

ermöglicht es, daß die Benutzerroutinen, die in den Operanden ERROPT bzw. WLRERR genannt sind, logisch mit dem ERET-Makro enden können.

Die Angabe des Operanden ERROPT ist freigestellt. Wenn er niedergeschrieben wird, so ist durch ihn die Behandlung fehlerhafter Sätze festgelegt. IGNORE bewirkt, daß die fehlerhaften Sätze so zur Verfügung gestellt werden, als ob sie frei von Fehlern

sind, d. h. sie werden genau so wie alle anderen Sätze behandelt. SKIP hat zur Folge, daß alle Sätze eines fehlerhaften Blockes übergangen werden; die Verarbeitung fährt mit dem nächsten Block fort. Wird bei ERROPT durch name eine Benutzerroutine genannt, so wird dieser beim Auftreten eines Fehlers die Programmablaufsteuerung übertragen. Fehlt nun der Operand ERREXT in der Dateidefinition, so enthält Register 1 die Adresse des fehlerhaften Blockes und Register 14 die Rücksprungadresse. Nach der Rückkehr wird der fehlerhafte Block übersprungen. Bei Angabe von ERREXT enthält Register 1 jedoch die Adresse einer Argumentenliste. Das erste Wort dieser Liste enthält die Adresse der Adresse der Dateidefinition DTFMT, das zweite Wort die Adresse des fehlerhaften Blockes.

Ist der Operand WLRERR nicht angegeben, so wird die ERROPT-Routine auch zur Behandlung von Sätzen falscher Länge herangezogen. Der Operand WLRERR bezieht sich ausschließlich auf Eingabesätze. Die Register werden für die bei WLRERR genannte Routine in genau derselben Weise gesetzt wie für die bei ERROPT genannte Routine

Wenn geblockte Sätze variabler Länge im Ausgabegebiet zusammengestellt werden, wird durch VARBLD das Register bezeichnet, daß nach jedem PUT (und auch nach dem OPEN) aufdatiert wird; es verweist auf die Anzahl der noch im Ausgabegebiet zur Verfügung stehenden Bytes. Die Ausführung des Makros TRUNC sollte immer dann veranlaßt werden, wenn der nächste Satz nicht mehr in den verbleibenden Restteil des Ausgabegebietes hineinpaßt. Im allgemeinen benutzt man bei solchen Dateien ein Arbeitsgebiet.

Der Operand NOTEPNT kann nur bei Arbeitsdateien eingesetzt werden.

Aufgabengruppe 57

1) Schreibe das Programm der Aufgabengruppe 56, Aufgabe 2 neu. Benutze dabei die Dateidefinitionen DTFCD und DTFPR sowie die EA-Makros.

2) Schreibe das Programm neu, das im Abschnitt 20.3. vorgestellt ist. Benutze dabei die EA-Makros sowie geeignete Dateidefinitionen (DTF-Statements).

3) Schreibe ein Programm in der AL, das eine Tabelle der Quadratzahlen und der Kubikzahlen in der folgenden Form druckt:

N	QUADRATZ.	KUBIKZAHL
1Ø	Ø1ØØ	ØØ1ØØØ
11	Ø121	ØØ1331
.		
:		
99	98Ø1	97Ø299

4) Schreibe ein Programm in der AL, das Karten einliest, die auf den Spalten 1 bis 5 die Zahl X enthalten, und das die Werte von X und von 2X + 1 druckt!

5) Schreibe ein Programm, das Karten einliest, bei denen auf den Spalten 1 bis 8 ein Wort abgelocht ist, das aus beliebigen alphamerischen Zeichen besteht. In Abhängigkeit vom ersten Zeichen des eingelesenen Wortes ist eine der folgenden Nachrichten zu drucken:

a) DER ERSTE BUCHSTABE VON xxxxxxxx IST EIN VOKAL.
b) DER ERSTE BUCHSTABE VON xxxxxxxx IST EIN KONSONANT.
c) DAS ERSTE ZEICHEN VON xxxxxxxx IST EINE ZIFFER.
d) DAS ERSTE ZEICHEN VON xxxxxxxx IST EIN SONDERZEICHEN.

Hierbei bedeutet das xxxxxxxx eingelesene Wort.

6) Welche Druckausgabe wird durch das Programm veranlaßt, das in der nachstehenden Abbildung vorgestellt wird?

Die Eingabewerte in den Spalten 1 bis 5 betragen der Reihe nach

a) ØØØØØ
b) ØØ123
c) ØØ12K
d) 99999

Nimm dabei an, daß Überlaufunterbrechungen nicht markiert sind. Das EOJ-Makro gibt die Programmablaufsteuerung an den Supervisor zurück.

```
R       BALR  11,0
        USING *,11
        OPEN  XEIN,YAUS
S       GET   XEIN
        PACK  X,XU(5)
        ZAP   Y,EINS
        AP    Y,X
        AP    Y,X
        UNPK  YU,Y
        PRTOV YAUS,12
        PUT   YAUS
        B     S
E       CLOSE XEIN,YAUS
        EOJ
XEIN    DTFCD BLKSIZE=80,DEVADDR=SYSRDR,EOFADDR=E,IOAREA1=XU
YAUS    DTFPR BLKSIZE=7,DEVADDR=SYSLST,IOAREA1=YU,PRINTOV=YES
XU      DS    CL80
YU      DS    CL7
X       DS    CL3
Y       DS    CL4
EINS    DC    PL1'-1'
        END   R
```

7) Ändere das Programm von Aufgabe 6) so ab, daß die Ausgabezeile die Werte von X und Y enthält; zwischen diesen beiden Werten sollen zwei Leerstellen verbleiben.

8) Das Programm von Aufgabe 7) ist wie folgt zu modifizieren:

Bei der Ausgabe sollen alle führenden Nullen unterdrückt werden; außerdem soll ein evtl. vorhandenes Minuszeichen unmittelbar vor der ersten gültigen Ziffer gedruckt werden.

9) Schreibe ein Programm, das den Wert von x von den Spalten 1 bis 10 einer Karte einliest und das sowohl den x-Wert als auch den y-Wert in eine Karte stanzt (der y-Wert soll auf den Spalten 11 bis 20 erscheinen). Der x-Wert ist ohne Dezimalpunkt abgelocht; die letzten acht Stellen sollen als Dezimalstellen interpretiert werden. Der y-Wert ist auf sieben Dezimalstellen zu runden, er soll ebenfalls ohne Dezimalpunkt

gestanzt werden (drei Stellen vor dem Komma, sieben nach dem Komma). Der Wert von y soll aus der nachstehenden Funktionsgleichung ermittelt werden:

$$y = \begin{cases} 70 - x & \text{für } x \geqq 0 \\ 2x^2 - 64 & \text{für } 0 < x \leqq x \leqq 90 \\ x - 100 & \text{für } x > 90 \end{cases}$$

Es sei noch einmal betont, daß sowohl der x-Wert als auch der y-Wert ohne Dezimalpunkt gestanzt werden sollen.

21.8. Die Verwendung von Pseudoabschnitten DSECT

In diesem Abschnitt wollen wir uns mit der Programmierung des folgenden Problems befassen: Es ist ein Kartenpaket einzulesen und der Inhalt der einzelnen Karten ist als Satz auf ein Magnetband zu schreiben. Wir wollen beim Magnetband den Blockungsfaktor 4 benutzen. Die Karteninhalte sollen als 80-stellige Sätze aufgezeichnet werden, wobei die Spalten 73 bis 80 die ersten acht Stellen und die Spalten 1 bis 72 die letzten 72 Stellen des Magnetbandsatzes belegen sollen.

Wir schreiben zuerst das Programm in der folgenden Fassung: Wir benutzen ein Eingabegebiet, ein Arbeitsgebiet für die Eingabe, ein Ausgabegebiet und ein Arbeitsgebiet für die Ausgabe (Abb. 21.9.).

Man beachte, daß E30 und E40, die beiden Puffergebiete, im Programm definiert sind und daß sie dem Verwendungszweck entsprechen müssen, der für sie in den Dateidefinitionen vorgesehen wurde. Im übrigen bezieht sich das Problemprogramm niemals auf

```
*                P R O G R A M M  1
*                ==================
*
*                PROGRAMM MIT ARBEITSGEBIETEN
*
CDTE      CSECT
KARTIN    DTFCD  TYPEFLE=INPUT,DEVICE=3505,DEVADDR=SYSIPT,EOFADDR=E20,
                 BLKSIZE=80,IOAREA1=E30,WORKA=YES
BANDAU    DTFMT  TYPEFLE=OUTPUT,DEVADDR=SYS004,RECFORM=FIXBLK,
                 FILABL=NO,BLKSIZE=320,RECSIZE=80,WORKA=YES,
                 IOAREA1=E40
E10       BALR   11,0
          USING  *,11
          OPEN   KARTIN,BANDAU
E11       GET    KARTIN,E31
          MVC    E51,E31+72
          MVC    E52,E31
          PUT    BANDAU,E50
          B      E11
E20       CLOSE  KARTIN,BANDAU
          EOJ
E30       DS     CL80            EINGABE-PUFFER
E31       DS     CL80            EINGABE-ARBEITSGEBIET
E50       DS     0CL80           AUSGABE-ARBEITSGEBIET
E51       DS     CL8
E52       DS     CL72
E40       DS     CL320           AUSGABE-PUFFER
          END    E10
```

Abb. 21.9. Programm 1: Arbeitsgebiete bei der Eingabe und bei der Ausgabe

ihre Inhalte; da sie Puffergebiete sind, werden sie nur von den EA-Operationen berührt, die zur gleichen Zeit wie die Verarbeitung im Hauptprogramm ablaufen.

Die Verzweigung nach E20 erfolgt automatisch und ist in der Tat eine mehr oder weniger unabhängige Routine. Das EOJ-Makro wird zur Beendigung des Programmes verwendet; es gibt die Programmablaufsteuerung an den Supervisor zurück. Wir werden die Supervisormakros im Abschnitt 21.17. behandeln.

Interessant ist noch am Programm 1 (Abb. 21.9.), wie durch E50 ein achtzigstelliges Feld definiert wird. Durch die beiden folgenden Vereinbarungen wird dieses Feld anschließend in zwei Teilfelder unterteilt, und zwar einmal in einen achtstelligen Teil und zum anderen in einen 72-stelligen Teil. Dadurch können wir auf die Notierung der Längenoperanden in den MVC-Instruktionen verzichten. Wir hätten diese sonst wie folgt schreiben müssen:

```
MVC    E5Ø(8'),E31+72
MVC    E5Ø+8(72),E31
```

Bei der 2. Version dieses Programmes (Programm 2 in Abb. 21.10.) wollen wir für die Ausgabe das Arbeitsgebiet beibehalten, jedoch es für die Eingabe tilgen. Da es bei der Eingabe kein Arbeitsgebiet und nur ein Eingabegebiet gibt, können sich nur Ausgabe und Verarbeitung überlappen. Das Fehlen der Überlappung zwischen dem Lesen und anderen Operationen zieht es nach sich, daß diese Version eine viel längere Ausführungszeit benötigt. Dabei sparen wir am Hauptspeicherplatz gerade 80 Bytes ein. Da ein Magnetband sehr viel schneller durchläuft als Karten eingezogen werden können, würde es weitaus sinnvoller sein, wenn wir die Überlappung bei der Bandausgabe eliminieren und stattdessen ein Arbeitsgebiet bei der Eingabe einrichten, wodurch diese mit der Verarbeitung überlappt werden kann.

```
*              P R O G R A M M  2
*              ==================
*
*              PROGRAMM OHNE ARBEITSGEBIET BEI DER EINGABE
*
CDTE     CSECT
KARTIN   DTFCD  TYPEFLE=INPUT,DEVADDR=SYSIPT,EOFADDR=E20,
                BLKSIZE=80,IOAREA1=E30
BANDAU   DTFMT  TYPEFLE=OUTPUT,DEVADDR=SYS004,RECFORM=FIXBLK,
                FILABL=NO,BLKSIZE=320,RECSIZE=80,WORKA=YES,
                IOAREA1=E40
E10      BALR   11,0
         USING  *,11
         OPEN   KARTIN,BANDAU
E11      GET    KARTIN
         MVC    E51,E30+72
         MVC    E52,E30
         PUT    BANDAU,E50
         B      E11
E20      CLOSE  KARTIN,BANDAU
         EOJ
E30      DS     CL80          EINGABE-PUFFER
E50      DS     OCL80         AUSGABE-ARBEITSGEBIET
E51      DS     CL8
E52      DS     CL72
E40      DS     CL320         AUSGABE-PUFFER
         END    E10
```

Abb. 21.10. Programm 2: Arbeitsgebiet bei der Ausgabe

In unserer dritten Version (Programm 3 in Abb. 21.11.) richten wir deshalb das Arbeitsgebiet für die Karteneingabe wieder ein und streichen dafür das Arbeitsgebiet bei der Magnetbandausgabe.

```
*                P R O G R A M M  3
*                ==========*======
*
*                PROGRAMM OHNE ARBEITSGEBIET BEI DER AUSGABE
*
CDTF      CSECT
KARTIN    DTFCD TYPEFLE=INPUT,DEVADDR=SYSIPT,EOFADDR=E20,DEVICE=3505,    1
                BLKSIZE=80,IOAREA1=E30,WORKA=YES
BANDAU    DTFMT TYPEFLE=OUTPUT,DEVADDR=SYS004,RECFORM=FIXBLK,            1
                FILABL=NO,BLKSIZE=320,RECSIZE=80,IOREG=(3),            2
                IOAREA1=E40
E10       BALR  11,0
          USING *,11
          OPEN  KARTIN,BANDAU
E11       GET   KARTIN,E31
          USING CDTF,3
          MVC   E51,E31+72
          MVC   E52,E31
          PUT   BANDAU
          B     E11
E20       CLOSE KARTIN,BANDAU
          EOJ
CDTF      DSECT
E50       DS    0CL80
E51       DS    CL8
E52       DS    CL72
CDTF      CSECT
E30       DS    CL80          EINGABE-PUFFER
E31       DS    CL80          EINGABE-ARBEITSGEBIET
E40       DS    CL320         AUSGABE-PUFFER
          END   E10
```

Abb. 21.11. Programm 3: Arbeitsgebiet bei der Eingabe

Beim dritten Programm werden die Eingabedaten durch das GET-Makro vom Puffer in das Arbeitsgebiet übertragen. Während dort die Verarbeitung fortgesetzt wird, wird die nächste Karte gelesen. Dem Register 3 ist die Aufgabe zugewiesen, die Anfangsadresse des Bandsatzes festzuhalten, der gerade zusammengestellt wird. Durch das PUT-Statement wird entweder diese Adresse um 80 erhöht oder, wenn das Ausgabegebiet gefüllt ist, ein Bandblock geschrieben und das Register 3 auf den Ausgangswert zurückgesetzt. Damit wir uns auf das Register 3 beziehen können, verwenden wir die Vereinbarung

```
USING CDTF,3
```

Durch diese Vereinbarung wird dem Übersetzungsprogramm, d.h. dem Assembler, mitgeteilt, daß es voraussetzen kann, im Register 3 die Adresse des ersten Bytes von CDTF zu finden, d.h. die Adresse von E5Ø.

Die tatsächlichen Werte, die vom IOCS ins Register 3 gestellt werden, sind die folgenden:

A (E40)
A (E40 + 80)
A (E40 + 160)
A (E 40 + 240)

Es sollte noch einmal vermerkt werden, daß bei diesem Programm keine Überlappung zwischen der Bandausgabe und der Verarbeitung erfolgt.

Zum Schluß wollen wir eine vierte Version (Programm 4) in Abb. 21.12. betrachten, bei der für beide Dateien je zwei EA-Gebiete vorgesehen sind. Werden für eine Datei zwei EA-Gebiete bereitgestellt, so ist für sie die Angabe von Arbeitergebieten eigentlich überflüssig. Wir verzichten deshalb beim Programm 4 auf solche.

```
*          P R O G R A M M  4
*          ==================
*
*          PROGRAMM MIT ZWEI EA-GEBIETEN FUER JEDE DATEI
*
CDTE     CSECT
KARTIN   DTFCD  TYPEFLE=INPUT,DEVADDR=SYSIPT,EOFADDR=E20,                1
                BLKSIZE=80,IOREG=(2),IOAREA1=E30,IOAREA2=E31
BANDAU   DTFMT  TYPEFLE=OUTPUT,DEVADDR=SYS004,RECFORM=FIXBLK,            1
                FILABL=NO,BLKSIZE=320,RECSIZE=80,IOREG=(3),              2
                IOAREA1=E40,IOAREA2=E41
E10      BALR   11,0
         USING  *,11
         USING  CDTF,3
         USING  CDTG,2
         OPEN   KARTIN,BANDAU
E11      GET    KARTIN
         MVC    E51,E60+72
         MVC    E52,0(2)
         PUT    BANDAU
         B      E11
E20      CLOSE  KARTIN,BANDAU
         EOJ
E30      DS     CL80          EINGABEGEBIET 1
E31      DS     CL80          EINGABEGEBIET 2
E40      DS     CL320         AUSGABEGEBIET 1
E41      DS     CL320         AUSGABEGEBIET 2
CDTF     DSECT
E50      DS     OCL80
E51      DS     CL8
E52      DS     CL72
CDTG     DSECT
E60      DS     CL80
         END    E10
```

Abb. 21.12. Programm 4: Zwei EA-Gebiete bei der Eingabe und bei der Ausgabe

Auf das Karteneingabegebiet haben wir uns auf zwei Weisen bezogen: einmal haben wir über eine DSECT-Vereinbarung erreicht, daß eine bestimmte Registeradresse für E60 vergeben wird, zum anderen haben wir das entsprechende Register selbst direkt niedergeschrieben. Wenn wir die zweite Form der Bezugnahme für die beiden MVC-Instruk-

tionen benutzt hätten, hätten wir die DSECT-Vereinbarung für CDTG überhaupt nicht gebraucht. Bei vielen einfachen und übersichtlichen Fällen ist es in der Tat wesentlich einfacher, wenn man sich nicht mit der DSECT-Vereinbarung herumzuquälen braucht.

Im Register 2 stehen abwechselnd die Werte

A (E30)
und A (E31)

Im Register 3 wechseln sich dagegen acht verschiedene Werte ab, nämlich:

A (E40 + n*80) für n = 0, 1, 2, 3
und A (E41 + n*80) für n = 0, 1, 2, 3

Nach dem Wert A (E41 + 240) wiederholt sich der Zyklus von vorn.

Wenn ein durch die DSECT-Vereinbarung eingeleiteter Pseudoabschnitt übersetzt wird, wird er stets relativ zu Null übersetzt. Das mit ihm verknüpfte Register wird ihm außerhalb seiner eigenen Vereinbarungen zugewiesen. (Bei unserem Programm bewirken die GET- und PUT-Statements die Zuweisungen der Register). Bei der Übersetzung von Pseudoabschnitten wird diesen tatsächlich niemals Speicherplatz zugeordnet; eine solche Zuordnung muß schon in einem normalen Programmabschnitt erfolgen, d. h. in einem durch START bzw. CSECT eingeleiteten. In unserem Fall erfolgt die Zuordnung von Speicherplätzen durch die Definitionen E30, E31, E40 und E41.

Man kann kaum Gegenargumente ins Feld führen, wenn von uns die Feststellung getroffen wird, daß man im allgemeinen zur einfachsten Codierung gelangt, wenn man nur ein EA-Puffergebiet und dafür aber ein Arbeitsgebiet vorsieht. Diese Methode führt auch gewöhnlich zum schnellsten Programmablauf.

21.9. Die Dateidefinition DTFSD

Die Operanden, die für die Dateidefinitionen von starr fortlaufend organisierten Dateien gelten, die auf SEDZ liegen, sind in der Abb. 21.13. aufgeführt. Die Arbeitsdateien werden im Abschnitt 21.10. behandelt.

Im allgemeinen gleichen die Eintragungen, die bei der Dateidefinition DTFSD vorgenommen werden können, den Eintragungen, die für Dateien gleicher Organisationsform auf Magnetbändern gegeben werden können. Es gibt jedoch die folgenden Unterschiede:

1) Für geblockte Sätze fester Länge kann das Makro TRUNC benutzt werden, um einen verkürzten Block aufzuzeichnen. In bestimmten Fällen kann dadurch die Ausnutzung der Plattenspuren verbessert werden.

2) Die Eintragung

   VERIFY=YES

   bewirkt, daß alle Schreiboperationen einer Prüfung unterzogen werden. Diese Prüfung beinhaltet das Lesen (mit dem Kennzeichen für SKIP) jedes geschriebenen Satzes unmittelbar nach dem Aufzeichnen. Dadurch ist eine gesonderte Plattenumdrehung immer dann erforderlich, wenn ein Satz geschrieben wurde. Die Prüfung wird jedoch gewöhnlich nicht eingesetzt.

3) Wenn die Eintragung

   UPDATE=YES

vorliegt, dürfen PUT-Makros auch für Eingabedateien gegeben werden. Diese Eigenschaft ist typisch für die Speichereinheiten mit direkter Zugriffsmöglichkeit (SEDZ). Wenn eine Datei gelesen und aufdatiert werden soll, so gibt der Programmierer für jeden Satz zunächst ein GET-Makro. Wenn der Satz ungeändert auf der SEDZ verbleiben soll, so braucht er bloß eine andere GET-Makroinstruktion hinauszuschicken und der nächste Satz wird gelesen. Wenn jedoch bei der Verarbeitung des gelesenen Satzes festgestellt wird, daß der Satz zu ändern oder zu ersetzen ist, so braucht der Programmierer für diesen Fall nur ein PUT-Makro zu geben. Wenn also ein PUT-Makro für eine Eingabedatei hinausgeschickt wird, wird einfach der augenblicklich in Bearbeitung befindliche Satz der SEDZ ersetzt. Die Verarbeitung wird freilich durch das Aufdatieren etwas verlangsamt, so daß dieser Operand nur dann benutzt werden sollte, wenn diese Maßnahme unumgänglich notwendig ist.

4) Die Eintragung

```
CONTROL=YES
```

ermöglicht die Verwendung des CNTRL-Makros. Dieses Makro besitzt die Form:

```
CNTRL dateiname,SEEK
```

Die Verwendung dieses Makros hat zur Folge, daß der Zugriffsmechanismus des Plattenspeichers so eingestellt wird, daß zum nächsten zu lesenden Satz zugegriffen werden kann. Wenn die Datei aus geblockten Sätzen besteht, so kommt diese Vorabeinstellung nur beim letzten Satz eines Blockes zur Geltung. Nach dem Lesen des letzten Satzes eines Blockes wird der Zugriffsmechanismus auf den nächsten zu lesenden Satzblock eingestellt. Der Vorteil der Verwendung dieses Makros liegt darin, daß die Bewegung des Zugriffsmechanismus nur gestartet wird; die Bewegung wird während der Zeit weitergeführt, in der der Computer mit der Verarbeitung fortfährt. Somit überlappen sich Suchen und Verarbeitung. Die Verwendung dieses Makros ist freilich nicht unbedingt notwendig; denn jedes GET-Makro bewirkt ebenfalls die Einstellung des Zugriffsmechanismus, wenn dieser sich nicht in der gewünschten Lage befindet. Somit besteht der einzige Vorteil des CNTRL-Makros darin, daß es Zeit einsparen hilft, jedoch kann das bei einem Mehrprogrammbetrieb auch schief gehen. Deshalb sollte dieses Makro generell vermieden werden.

21.10. Arbeitsdateien

Dateien, die durch

```
TYPEFLE=WORK
```

in einer der beiden Dateidefinitionen DTFMT oder DTFSD gekennzeichnet sind, werden als Arbeitsdateien bezeichnet. Arbeitsdateien, die auf Plattenspeichern liegen, besitzen Standardkennsätze[1], Arbeitsdateien auf Magnetbändern können dagegen Standardkennsätze haben, sie brauchen es jedoch nicht, d.h. sie weisen im allgemeinen keine Kennsätze auf. Wenn Benutzerkennsätze vorhanden sind, so werden diese nicht bearbeitet. Nachsätze werden für Arbeitsdateien weder geschrieben noch bearbeitet.

Eine Arbeitsdatei ist dazu bestimmt, während des Programmablaufes als Hilfsspeicher zu dienen. Als solche unterscheidet sie sich deshalb grundlegend von einer EA-Datei.

---

1) Außerdem werden zur Identifizierung der Datei die Steuerkarten DLBL und EXTENT benötigt. Jede solche Datei ist auf einen Plattenstapel beschränkt, doch kann sie mehrere Bereiche aufweisen.

Operand	*Bedeutung*	*Bemerkungen*	
BLKSIZE=n	Länge des EA-Gebietes	Für jede Datei erforderlich	*
CONTROL=YES	Verwendung des Makros CNTRL	--	
DELETFL=NO	Bei CLOSE keine Streichung (Zerstörung) der Datei	Nur für Arbeitsdateien gültig	*
DEVADDR=SYSnnn	Festlegung des logischen Gerätenamens	Erforderlich, wenn bei den Steuerkarten EXTENT kein SYSnnn-Operand vorhanden ist	*
DEVICE={2311 / 2314 / 2321 / 333Ø / 334Ø}	Festlegung des Gerätetyps	Für alle Geräte außer IBM 2311 erforderlich	*
EOFADDR=name	Verzweigungsadresse bei Dateiende	Für Eingabedateien und Arbeitsdateien gültig	
ERREXT=YES	Verwendung des Makros ERET	--	*
ERROPT={IGNORE / SKIP / name}	Verarbeitung von fehlerhaften Sätzen	SKIP gilt bei Ausgabedateien nicht	*
FEOVD=YES	Verwendung des Makros FEOVD		
IOAREA1=name	Eingabe/Ausgabe-Gebiet	Für jede Datei erforderlich; name ist ein Adreßausdruck	*
IOAREA2=name	2. Eingabe/Ausgabe-Gebiet	--	*
IOREG=(r)	Steuerung der EA-Gebiete bzw. Verarbeitung geblockter Sätze ohne Arbeitsgebiet	r ist eines der Register 2 bis 12; es enthält die Satzadresse	
LABADDR=name	Verarbeitung von Benutzerkennsätzen	--	
NOTEPNT={YES / POINTRW}	Bei YES: Verwendung der Makros NOTE, POINTR, POINTW und POINTS Bei POINTRW: Verwendung der Makros NOTE, POINTW und POINTR	Anwendung nur bei Arbeitsdateien	

Operand	*Bedeutung*	*Bemerkungen*	
RECFORM={FIXUNB / FIXBLK / VARUNB / VARBLK / SPNUNB / SPNBLK / UNDEF}	Satzform	Nicht erforderlich	
RECSIZE={n / (r)}	Satzlänge n oder Register r, das die Satzlänge enthält	Erforderlich bei FIXBLK und bei UNDEF; r ist eines der Register 2 bis 12	
TRUNCS=YES	Verwendung des Makros TRUNC bei Ausgabedateien bzw. eine Eingabedatei mit geblockten Sätzen fester Länge enthält Kurzblöcke	--	
TYPEFLE={INPUT / OUTPUT / WORK}	Dateiverwendung	Bei Eingabedateien nicht erforderlich	*
UPDATE=YES	Aufdatierung bei Eingabedateien oder bei Arbeitsdateien	--	
VARBLD=(r)	Geblockte Sätze variabler Länge werden im Ausgabegebiet zusammengestellt; kein Arbeitsgebiet	r ist eines der Register 2 bis 12	
VERIFY=YES	Schreibprüfung	--	*
VARBLD=(r)	Zusammenstellung geblockter Ausgabesätze variabler Länge im Ausgabegebiet	r ist eines der Register 2 bis 12	
WLRERR=name	Verarbeitung von Sätzen falscher Länge bei einer Eingabedatei	--	
WORKA=YES	Verwendung von Arbeitsbereichen bei GET- oder PUT-Makros	--	

Anmerkung: Die mit einem Stern versehenen Operanden gelten auch für Arbeitsdateien

Abb. 21.13. Mögliche Operanden bei der Dateidefinition DTFSD

Sie wird mehr oder weniger als eine Erweiterung des Hauptspeichers der Zentraleinheit betrachtet. Ihr Inhalt wird als ein zeitweiliger Inhalt angesehen; im allgemeinen wird er am Programmende zerstört werden, d. h. nicht mehr zur Verfügung stehen. Arbeitsdateien werden stets sowohl ausgegeben als auch eingegeben.

Das OPEN-Statement ist dann und nur dann erforderlich, wenn Kennsätze zu bearbeiten sind. Das CLOSE-Statement ist bei Bändern nicht erforderlich, doch ist es recht nützlich, weil es das Zurückspulen des betreffenden Bandes bewirkt. Bei Plattenspeichern bewirkt dieses Statement die Entfernung der Dateieintragung aus dem Datenträgerinhaltsverzeichnis, es sei denn, daß diese Maßnahme durch den Operanden DELETFL=NO verhindert wird. — Bei Magnetbändern bewirkt das erste OPEN-Statement das Zurückspulen des Bandes und das Prüfen der Kennsätze. Wird ein Standardkennsatz gefunden, so wird er durch einen ersetzt, der aus der Zeichenfolge HDR1, gefolgt von 76 Leerzeichen, besteht. Im anderen Fall wird eine Bandmarke aufgezeichnet. Die Steuerkarte TLBL ist nicht erforderlich. Das OPEN-Statement ist nicht erforderlich bei einer auf einem Magnetband hängenden Arbeitsdatei. Eine solche Datei kann anfangs auch nicht als Eingabedatei herangezogen werden. Das CLOSE-Statement ist bei Dateien auf Magnetbändern nicht erforderlich, doch zeichnet es eine Bandmarke auf und spult anschließend das Band zurück. Wenn nun im Programm ein zweites OPEN-Statement für die betreffende Datei gegeben wird, wird das Band so positioniert, daß der erste Satz gelesen werden kann; eine zweite Kennsatzverarbeitung findet jedoch nicht statt.

Die üblichen GET- und PUT-Makros sind für Arbeitsdateien nicht erlaubt. An ihrer Stelle sind die folgenden Makros einzusetzen:

```
READ   dn,SQ,gebiet[,länge]
WRITE  dn,SQ,gebiet[,länge]
WRITE  dn,UPDATE,gebiet[,länge]
CHECK  dn
NOTE   dn
POINTR dn,adresse
POINTW dn,adresse
POINTS dn
```

Hierbei bedeutet dn den Dateinamen der angesprochenen Datei. Die letzten vier Makros können nur dann verwendet werden, wenn in die Dateidefinition der Parameter NOTEPNT eingeschlossen ist. An Stelle des Dateinamens kann auch eine Registernotierung vorgenommen werden, d.h. es kann (r) an Stelle von dn stehen. Wenn eine solche Registernotierung angegeben werden soll, so muß zuvor die Adresse der DTF-Definition in das angegebene Register geladen werden. Unter gebiet ist das Gebiet im Hauptspeicher aufzuführen, in dem bzw. von dem die Daten übertragen werden[1]. Die Angabe des Gebietes kann entweder durch einen Adreßausdruck oder durch eine Registernotierung erfolgen. Sowohl bei dn als auch unter gebiet können die Register 2 bis 12 unbeschränkt gebraucht werden; außerdem kann bei dn auch das Register 1, bei gebiet auch das Register 0 angegeben werden. Die maximale Satzlänge wird aus dem Operanden BLKSIZE ermittelt, der ja bekanntlich zur Dateidefinition DTF gehört.

Bei ungeblockten Sätzen fester Länge kann der Programmierer die Angabe BLKSIZE = n ändern, indem er dem Halbwort mit der symbolischen Adresse dateiname L einen anderen Wert zuweist. Wenn beispielsweise die Datei mit OBST benannt ist und außerdem die neue Satzlänge im Register 8 steht, dann können wir wie folgt codieren, um die neue Länge in die Dateiparameter zu stellen:

```
STH    8,OBSTL
```

1) Es handelt sich hier also um ein Arbeitsgebiet

Wir haben also an den Dateinamen einfach ein L anzuhängen, um das Längenfeld ansprechen zu können.

Der Operand länge wird nur für Sätze mit undefinierter Länge benötigt, durch ihn ist die Länge des Satzes gegeben.

Der Längenoperand in den Makros READ und WRITE ist bei Dateien, die aus Sätzen undefinierter Länge bestehen, unbedingt erforderlich; es ist jedoch nicht gestattet, ihn für Dateien mit ungeblockten Sätzen fester Länge anzugeben. Wenn er verwendet wird, so darf er die Länge des EA-Gebietes direkt spezifizieren; es ist aber auch möglich, eine Registernotierung zu verwenden, in diesem Fall muß das betreffende Register die Länge enthalten. Beim READ-Makro ist es überdies auch möglich, ein S zu spezifizieren, wodurch die maximale Satzlänge unterstützt wird. Die maximale Satzlänge wird aus der Angabe BLKSIZE=n ermittelt. Der Hauptsinn in der Verwendung einer anderen Angabe als S besteht darin, daß man dafür sorgen will, daß nur der erste Satzteil in den Hauptspeicher eingelesen wird.

Beispiele von READ-Statements sind:

```
READ   FLNJ,SQ(3),8Ø
READ   KP78,SQ,AR45
```

Das erste Statement bezieht sich auf eine Datei mit Sätzen undefinierter Länge, das zweite auf eine Datei mit Sätzen fester Länge. Man beachte, daß für Arbeitsdateien IOAREA1 nicht angegeben wird, so daß beim zweiten Beispiel AR45 tatsächlich das Eingabegebiet ist.

Das Makrostatement WRITE besitzt die gleichen Operanden wie das READ-Statement; bei Benutzung eines Plattenspeichers kann jedoch der zweite Operand auch UPDATE sein. Wenn SQ spezifiziert wird, so wird ein neuer Satz mit Hilfe des Befehls „Schreiben der Kennzeichnung, des Schlüssels und der Daten (Write Count, Key, and Data)" aufgezeichnet. Wenn dagegen UPDATE spezifiziert wird, so wird stattdessen das Makro so aufgelöst, daß es zum Befehl „Schreiben der Daten (Write Data)" kommt; dadurch wird der vorhergehende Satzinhalt ersetzt. Bei Verwendung von UPDATE muß der neue Satz die gleiche Länge aufweisen wie der zu ersetzende alte Satz. Einem WRITE-Makro mit dem Operanden UPDATE geht gewöhnlich das zugehörige READ-Makro unmittelbar voraus. Für WRITE-Statements ist der Längenoperand S nicht erlaubt.

Durch die READ- und WRITE-Statements werden die EA-Operationen gestartet; bis zum Abschluß dieser Operationen verzögern sie das Programm nicht. Sie geben nach dem Starten sofort die Ablaufsteuerung an das Problemprogramm zurück. Mitunter, d.h. meist sogar kommt es zu einem Wartestatus, weil der Benutzer auf den Abschluß der EA-Operation warten muß. Das Makro

```
CHECK dateiname
```

bewirkt ein Warten bis zum Ende der laufenden Operation mit der spezifizierten Datei. Wenn die Operation beendet ist, wird die nächste Instruktion nach dem CHECK-Makro ausgeführt. Dazu kommt es jedoch dann nicht, wenn entweder die Dateiendebedingung (EOF) oder ein Fehler aufgetreten ist; in diesen Fällen wird der geeigneten Behandlungsroutine die Steuerung übergeben. Beim Schreiben einer Arbeitsdatei kann eine EOF-Bedingung ebenfalls auftreten, dann nämlich, wenn bei einem Magnetband das physikalische Ende des Bandes bzw. bei einem Plattenspeicher das Ende des vorgesehenen Bereiches erreicht ist. Somit muß im Gegensatz zu den gewöhnlichen Dateien für jede Arbeitsdatei eine EOF-Routine vorgesehen sein.

Das WRITE-Statement bewirkt den Beginn einer Ausgabeoperation. Auch bei ihr wird das CHECK-Makro benutzt, um auf die Beendigung der Operation zu warten. Die Satzlänge kann geändert werden, normalerweise bleibt sie freilich ungeändert.

Das Makro NOTE besitzt die folgende Form:

```
NOTE  dateiname
```

Es bewirkt, daß eine Satzkennzeichnung ins Register 1 gestellt wird. Bei Bändern besteht diese aus der Ziffernfolge 0bbb, wobei bbb ein Blockzähler ist, in dem die Anzahl der bisher geschriebenen oder gelesenen Blöcke festgehalten ist. Bei Plattenspeichern setzen sich die im Register 1 stehenden vier Bytes aus den Ziffern zkr zusammen, wobei z den Zylinder, k den Kopf und r die Satznummer des letzten gelesenen bzw. geschriebenen Satzes bedeuten. Im Anschluß an ein WRITE-Statement wird ins Register 0 eine Zahl gestellt, die die Anzahl der auf der betreffenden Spur verbleibenden Bytes angibt.

Das Makrostatement POINTS dient dazu, eine Einstellung (Positionierung) auf den Anfangspunkt einer Datei vorzunehmen. Liegt die Datei auf einem Magnetband, so wird dieses zunächst zurückgespult und anschließend wird über eine Bandmarke hinweggesprungen. Bei Plattenspeichern erfolgt eine Einstellung auf den ersten Satz des ersten vorgesehenen Plattenbereiches.

Bei den Makros POINTR und POINTW wird bei Magnetbanddateien durch den zweiten Operanden die Adresse eines vier Bytes umfassenden Feldes spezifiziert, in dem 0bbb steht. Bei auf Magnetplattenspeichern gelegenen Dateien ist durch adresse ein sechs Bytes langes Feld mit dem Inhalt zzkrll gestellt, wobei ll die Anzahl der auf der Spur verbleibenden Bytes bedeutet. Bei Magnetbändern erfolgt dann die Einstellung auf die vor dem Satz bbb gelegene Kluft, bei POINTW ist es die auf diesen Satz folgende Kluft. Bei Plattenspeichern sind POINTR und POINTW äquivalent. Beide positionieren die Datei so, daß ein nachfolgendes READ- oder WRITE-Statement mit dem Operanden SQ zum Satz mit der Plattenadresse zzkr zugreift. Ein WRITE-Statement mit dem Operanden UPDATE darf nicht sofort nach irgendeinem der Statements POINTS, POINTR oder POINTW gegeben werden, ihm muß ein READ-Statement vorangehen.

Die Makros NOTE und alle der POINT-Gruppe dürfen nach READ- und WRITE-Makros erst dann codiert werden, wenn ihnen ein CHECK-Makro vorausgeht. Man beachte, daß immerhin die Makros der POINT-Gruppe Direktverarbeitung auf Magnetbändern und Plattenspeichern ermöglichen. Damit steht uns ein wertvolles Programmierwerkzeug zur Verfügung, das jedoch mit einiger Vorsicht gehandhabt werden muß. Die IOCS-Routinen führen einige Prüfungen bezüglich der Gültigkeit der in diesen Makros enthaltenen Informationen durch. Unkorrekte Werte in den Makros der POINT-Gruppe können zu unkorrekten und unerwarteten Resultaten führen.

Ein Makro der POINT-Gruppe, dem ein READ-Makro folgt, braucht keine korrekten Längenwerte ll auf dem Plattenspeicher zu haben.

Wird der Versuch unternommen, einen Satz zu schreiben, der zu lang ist, um in den noch verfügbaren Platz auf einer Spur hineinzupassen, so wird der im letzten Teil der Spur stehende Satz nicht ersetzt. Der neue Satz wird stattdessen als erster Satz auf die Folgespur geschrieben. Bei einem Plattenspeicher wird durch das WRITE-Makro mit dem Operand SQ jede alte Information, die in der Spur auf den gerade geschriebenen Satz folgt, gelöscht. WRITE mit dem Operanden UPDATE ersetzt hingegen den zuletzt gelesenen Satz. Ein WRITE-Statement mit dem Operanden UPDATE nach einem WRITE-Statement mit dem Operanden SQ führt zu nicht vorhersagbaren Resultaten.

READ und WRITE-Statements können in beliebiger Weise miteinander vermischt werden. Bei Bändern sollte jedoch daran erinnert werden, daß bei einem WRITE die Sätze hinter dem gerade vorliegenden Punkt nicht greifbar sind.

21.11. Die Dateidefinition DTFDI

Eine beschränkte Form der Geräteunabhängigkeit wird durch Definition von Dateien mittels DTFDI erreicht. Für sie sind jedoch nur ungeblockte Sätze fester Länger erlaubt. Von 96spaltigen Lochkarten werden nur die ersten 80 Spalten gelesen. Die Makrostatements CNTRL und PRTOV sind nicht erlaubt. Arbeitsgebiete werden nicht unterstützt. Nur einige bestimmte logische Systemdateien können benutzt werden. Die Standardsatzlängen sind gleich den Maximalwerten, die für die einzelnen Gerätetypen zulässig sind. Ein Steuerzeichen nach ASA wird bei SYSPCH- und SYSLST-Dateien angenommen.

Mit DTFDI definierte Dateien dürfen allen Dateien zugewiesen werden, die vom Betriebssystem DOS für die spezielle logische Systemeinheit zulässig sind.

Die für DTFDI zulässigen Operanden sind in der Abb. 21.14. zusammengestellt.

Operand	*Bedeutung*	*Bemerkungen*
DEVADDR={SYSIPT / SYSPCH / SYSLST / SYSRDR}	Festlegung des logischen Gerätenamens	Für jede Datei erforderlich
EOFADDR=name	Verzweigungsadresse bei Dateiende	--
ERROPT={IGNORE / SKIP / name}	Behandlung von EA-Fehlern	--
IOAREA1=name	EA-Gebiet	Für jede Datei erforderlich
IOAREA2=name	Zweites EA-Gebiet	--
IOREG=(r)	Bei zwei EA-Gebieten: Register zur Steuerung der EA	r = 2 bis r = 12; sie enthalten die Adresse des EA-Gebietes, wenn deren zwei benutzt werden. Standard für r ist 2.
RECSIZE=n	Satzlänge	Standardlängen sind: 121 für SYSLST, 81 für SYSPCH, 80 für SYSIPT und SYSRDR
WLRERR=name	Verarbeitung von Eingabesätzen mit falscher Länge (name ist dabei der Name der Eingangsstelle in die entsprechende Routine)	Bei Weglassung dieses Operanden werden die fehlerhaften Sätze so behandelt, wie es bei ERROPT angegeben wurde

Abb. 21.14. Mögliche Operanden bei der Dateidefinition DTFDI

Aufgabengruppe 58

1) Schreibe das Programm von Aufgabe 6 der Aufgabengruppe 57 neu! Verwende für die beiden Dateien die Dateidefinition DTFDI. Nimm an, daß die Druckerformulare 66 Zeilen pro Seite umfassen, 60 Zeilen zum Bedrucken und 6 Zeilen bleiben zwischen den Seiten (je 3 oben und unten auf jeder Seite) frei.

2) Schreibe ein Programm, mit dessen Hilfe Karten eingelassen und als 80-stellige ungeblockte Sätze auf ein Magnetband geschrieben werden können. Auf das Magnetband sollen Standardkennsätze aufgezeichnet werden.

3) Schreibe ein Programm, mit dessen Hilfe das Magnetband gelesen werden kann, das durch das Programm von Aufgabe 2) erstellt wurde. Die gelesenen Magnetbandsätze sind starr fortlaufend auf einen Plattenspeicher zu schreiben.

4) Ändere die Programme der Aufgaben 2) und 3) so ab, daß auf dem Magnetband die Sätze mit dem Blockungsfaktor 4 und auf dem Plattenspeicher mit dem Blockungsfaktor 5 gespeichert werden.

5) Auf einem Magnetband sind ungeblockte Kundensätze aufgezeichnet. Die Kundensätze sind nach aufsteigenden Kundennummern sortiert. Jeder Satz enthält die folgenden Informationen:

   Stelle 1 : 1
   Stellen 2 bis 9 : Kundennummer
   Stellen 10 bis 90 : Name und Adresse des Kunden
   Stellen 91 bis 100 : Guthaben oder Schuld des Kunden

   Ein Kundenguthaben ist durch eine neg. Zonung, d.h. D (hex) auf der Einerstelle gekennzeichnet.

   Die Umsätze sind auf Karten abgelocht, die nach aufsteigenden Kundennummern sortiert sind. Die Karten sind wie folgt eingeteilt:

   Spalte 1 : 2
   Spalten 2 bis 9 : Kundennummer
   Spalten 10 bis 19 : Betrag
   Spalte 20 : a) Lochung in Zeile 11 bei Belastungen
   b) Lochung in Zeile 12 bei Einzahlungen

   Für jeden Kunden können beliebig viele Umsatzkarten vorliegen; es braucht jedoch auch keine vorhanden zu sein.

   Schreibe ein Programm, durch das die Karten eingelesen und die Kundensätze auf den neuesten Stand gebracht (aufdatiert) werden können. Die Kontenstände der Kunden sind um die in den Karten ausgewiesenen Beträge zu berichtigen.

6) Das Programm zu Aufgabe 5) ist wie folgt zu ergänzen:

   a) Es ist für alle Kunden mit einem Kontenstand von 10000,— DM und mehr eine Karte zu stanzen. Die 80 Kartenspalten sollen dabei den Inhalt der Stellen 1 bis 70 sowie 91 bis 100 des auf dem Magnetband stehenden Kundensatzes enthalten.

   b) Von allen Kunden mit einem Kontenstand von 100000,— DM und mehr ist der Kundensatz zu drucken.

7) Es soll vorausgesetzt werden, daß die Kundendatei - siehe Aufgabe 5) - einen Schlußsatz enthält, der die Summe der Kontenstände aller Kunden auf den Stellen 89 bis 100

ausweist. Dieser Summensatz steht unmittelbar vor dem Dateiendesatz (EOF) und ist auf der Stelle 1 durch die Kennziffer 3 gekennzeichnet.
Das Programm von Aufgabe 5) bzw. Aufgabe 6) ist so zu erweitern, daß nach Abschluß der Verarbeitung auch der Summensatz auf den neusten Stand gebracht (aufdatiert) wird.

8) Ändere das Programm von Aufgabe 6) so ab, daß an Stelle des Stanzens von Karten ein Magnetband beschrieben wird. Während des Zurückspulens des Bestandsbandes (Kundenbandes) sind dann die Sätze von dem erstellten Arbeitsband herunterzudrucken.

9) Das Programm von Aufgabe 8) ist so abzuändern, daß an Stelle einer Bandarbeitsdatei eine Plattenspeicherarbeitsdatei verwendet werden kann.

10) Das Programm von Aufgabe 5) ist wie folgt abzuändern: Die Kundensätze sollen sich auf einem Plattenspeicher befinden und direkt durch das Programm auf den neuesten Stand gebracht werden.

21.12. Die Dateidefinition DTFDA

In der Abb. 21.15. sind die Operanden tabelliert, die bei Definitionen von regional organisierten (auch als gestreut bzw. direkt organisierte Dateien bezeichnet) verwendet werden können. Die Makros OPEN, CLOSE und LBRET werden bei einer regional organisierten Datei in der gleichen Weise benutzt wie bei einer starr fortlaufend (konsekutiv) organisierten Datei. Neben diesen Makros können außerdem die folgenden Makros eingesetzt werden:

```
READ   dn,KEY
READ   dn,ID
WRITE  dn,KEY
WRITE  dn,ID
WRITE  dn,AFTER
WRITE  dn,AFTER,EOF
WRITE  dn,RZERO
WAITF  dn
```

Anstelle der Niederschrift des Dateinamens dn kann man auch eine Registernotierung einsetzen, d.h. der erste Operand kann (r) oder (1) sein. Unter r ist hierbei eines der Register 2 bis 12 zu verstehen. Der Zugriff zur Datei kann mittels der physischen Kennzeichnung, der relativen dezimalen oder der relativen redezimalen erfolgen. Wird DSKXTNT weggelassen, so wird die physische Kennzeichnung erwartet und unter SEEKADR ist der Name eines acht Bytes langen Feldes zu codieren, das den üblichen Adreßverweis enthält (MBBZZKKR — Datenträger, Zelle, Zylinder, Kopf, Satz). Wenn dagegen DSKXTNT angegeben ist, dazu außerdem RELTYP=DEC, so stellt name unter SEEKADR den Namen eines zehnstelligen Dezimalfeldes mit Zonen dar, das TTTTTTTTRR (Spur; Satz) enthält. Ist DSKXTNT zusammen mit RELTYPE=HEX definiert, so stellt name den Namen eines vier Bytes umfassenden Feldes dar, das TTTR (Spur, Satz) enthält.

Wenn der Operand KEYLEN weggelassen wird, so werden keine Schlüssel verarbeitet; besitzen die Sätze Schlüssel, so werden diese weder gelesen noch zurückgeschrieben. Fehlt KEYARG, so besitzen die Sätze keine Schlüssel. Die Angabe unter SEEKADR wird dann zur Identifizierung von Spur und Satz verwendet. Ist dabei KEYARG ebenfalls vorhanden, so weisen die Sätze Schlüssel auf. Wird KEY in READ- oder WRITE-Statements

Operand	*Bedeutung*	*Bemerkung*
AFTER=YES	Verwendung von AFTER oder RZERO beim Makro WRITE	--
BLKSIZE=n	Länge des EA-Gebietes	Angabe für jede Datei erforderlich
CONTROL=YES	Verwendung des Makros CNTRL	--
DEVADDR=SYSnnn	Festlegung des logischen Gerätenamens	Erforderlich, wenn der Operand SYSnnn auf der Steuerkarte EXTENT fehlt. Diese Eintragung wird vernachlässigt, wenn die Steuerkarte den entsprechenden Operanden besitzt
DEVICE={2311 / 2314 / 2321 / 333Ø / 334Ø}	Festlegung des Gerätetyps	Erforderlich nur, wenn ein anderes Gerät als die IBM 2311 angesprochen werden soll
DSKXTNT=n	Dateien mit relativer Kennzeichnung	n Anzahl der Bereiche
ERRBYTE=name	Verwendung eines 2 Bytes langen Feldes für Fehlerschlüssel	Erforderlich für jede Datei
ERREXT=YES	Verwendung des Makros ERET	Zeigt nichtdatengebundene Übertragungsfehler unter ERRBYTE an
FEOVD=YES	Verwendung des Makros FEOVD	--
IDLOC=name	Verwendung der Kennzeichnung (5 Byte langes Feld)	--
IOAREA1=name	EA-Gebiet	--
KEYARG=name	Bezugnahme auf die Sätze durch ihre Schlüssel	--
KEYLEN=n	Schlüsselfeldlänge	Erforderlich bei Sätzen mit Schlüsseln
LABADDR=name	Verarbeitung von Benutzerkennsätzen	--
READID=YES	Angabe der Kennzeichnung beim READ-Makro	--

Operand	*Bedeutung*	*Bemerkungen*
READKEY=YES	Angabe des Schlüssels beim READ-Makro	--
RECFORM={FIXUNB / VARUNB / SPNUNB / UNDEF}	Satzform	Erforderlich für jede Datei
RECSIZE=(r)	Satzlänge bei Dateien mit Sätzen undefinierter Länge	Unter r ist eines der Register 2 bis 12 aufzuführen; dieses enthält die Satzlänge
RELTYPE={DEC / HEX}	Form der relativen Adressierung	--
SEEKADR=name	Feld für die Bezugnahme auf die Spuren (8 Bytes lang)	Erforderlich für jede Datei
SRCHM=YES	Durchsuchen aufeinanderfolgender Spuren (Mehrspuroperation)	Im READ- bzw. WRITE-Makro muß KEY angegeben sein
TRLBL=YES	Verarbeitung von Nachsätzen	Erforderlich ist LABADDR; nur für Ausgabe
TYPEFLE={INPUT / OUTPUT}	INPUT: Lesen und Prüfen von Kennsätzen OUTPUT: Schreiben von Kennsätzen	Für jede Datei erforderlich
VERIFY=YES	Schreibprüfung (bei der IBM 2321 erfolgt diese stets)	Nur bei Ausgabedateien möglich
WRITEID=YES	Verwendung der Kennzeichnung beim Makro WRITE	--
WRITEKY=YES	Verwendung des Schlüssels beim Makro WRITE	--
XTNTXIT=name	Verarbeitung von Bereichssteuerstatements der Jobsteuersprache	--

Abb. 21.15. Mögliche Eintragungen bei der Dateidefinition DTFDA

angegeben, dann wird die unter SEEKADR stehende Suchadresse nur zur Identifizierung der Spur benutzt, der Wert unter KEYARG dient dann zur Lokalisierung des Satzes.

Soll das Statement READ mit dem Operanden KEY benutzt werden, so muß

```
READKEY=YES
```

aufgeführt sein. Wird das READ-Statement mit ID eingesetzt, so ist

```
READID=YES
```

erforderlich. Bei Benutzung des WRITE-Statements mit dem Operanden KEY ist

```
WRITEKY=YES
```

und bei Benutzung von WRITE mit ID

```
WRITEID=YES
```

niederzuschreiben. Für die READ- und WRITE-Statements mit Schlüsselbezugnahme (KEY-Operand) ist zusätzlich die Niederschrift des Operanden KEYARG erforderlich. Die Angabe von

```
AFTER=YES
```

erlaubt die Benutzung des Makros WRITE mit den Operanden AFTER, AFTER, EOF und RZERO.

Liegt

```
AFTER=YES
```

vor und ist unter KEYLEN ein von Null verschiedener Wert angegeben, so sind alle Formen der Lese- und Schreibstatements erlaubt und im unter IOAREA1 spezifizierten Feld sind Kennzeichnung, Schlüssel und Daten des Satzes zu finden. Das EA-Gebiet muß also eine Länge aufweisen, die gleich der Summe aus 8, Schlüssellänge und Datenlänge ist. Bei

```
AFTER=YES
```

jedoch einer 0 unter KEYLEN oder gar bei Weglassung von KEYLEN enthält das EA-Gebiet nur die Kennzeichnung und die Daten. Alle Makros außer READ und WRITE mit dem Operanden KEY sind gestattet. Sind sowohl AFTER als auch KEYLEN weggelassen (oder ist KEYLEN gleich 0), so ist nur die Benutzung von READ und WRITE mit dem Operanden ID zulässig und im Gebiet, das unter EA spezifiziert ist, stehen nur die Daten. Ist der Wert bei KEYLEN positiv und AFTER fehlt, so enthält das EA-Gebiet nur die Daten, wenn außerdem weder READID noch WRITEID vorhanden sind. Sind dagegen READID oder WRITEID oder gar beide in die Definition der Datei eingeschlossen, so stehen im EA-Gebiet Schlüssel und Daten, die Kennzeichnung fehlt dann naturgemäß.

Das Makro READ verursacht das Suchen nach der unter SEEKADR angegebenen Spur. Bei Vorhandensein des Operanden ID wird danach der Satz gelesen, auf den im Satzteil der Suchadresse verwiesen wird. Beim Operanden KEY ist das anders: Hier wird der Satz gelesen, dessen Schlüssel unter KEYARG angegeben ist. Daten, Schlüssel und Daten oder Kennzeichnung, Schlüssel und Daten werden dabei so gelesen, wie wir es im vorigen Paragraphen beschrieben haben. Das READ-Statement startet nur die Leseoperation. Das WAITF-Makro wird dazu benutzt, um alle weiteren Operationen zu unterdrücken, bis das Lesen abgeschlossen ist. Bei der Ausführung von WAITF werden dabei anormale, während der Ausführung aufgetretene Bedingungen in ein zwei Bytes langes Feld mit Namen ERRBYTE gestellt, bevor die Programmablaufsteuerung zur nächstfolgenden Instruktion übergeht.

Bei der Ausführung des Makrostatements WRITE mit dem Operanden ID wird zunächst ein Suchen zu der Spur aufgesetzt, die unter SEEKADR vermerkt ist. Danach wird der Satz ersetzt, auf den im Satzteil der Suchadresse verwiesen wird; Schlüssel (falls vorhanden) und Daten werden durch den Inhalt von IOAREA1 ersetzt. Bei Ausgabe des Operanden KEY wird die Spur aufgesucht, auf die unter SEEKADR verwiesen wird. Danach geht es weiter zum Satz, der mit dem Wert unter KEYARG angesprochen ist; Schlüssel und Daten werden ersetzt durch den Inhalt von IOAREA1.

Die restlichen drei Varianten des WRITE-Makros können wie folgt beschrieben werden:

1) WRITE dn,RZERO
   Zunächst wird der Arm auf die Spur eingestellt, auf die SEEKADR verweist. R_0 wird mit ZZKKRLLb (Zylinder, Kopf, Satz, verfügbare Bytes) neu beschrieben. Dabei wird R auf 0 und LL auf die maximale Spurkapazität gesetzt. Im Anschluß an das Schreiben von R_0 wird der Rest der Spur gelöscht. Dieses Makro kann dazu verwendet werden, um einen Bereich auf dem Plattenspeicher zu initialisieren.

2) WRITE dn,AFTER
   Der Zugriffsmechanismus wird auf die durch SEEKADR spezifizierte Spur eingestellt und R_0 gelesen. Wenn noch genügend Platz auf der Spur vorhanden ist, um den Satz aufzuzeichnen (wobei zum Längenvergleich der Wert unter RECSIZE herangezogen wird), wird zunächst R_0 aufdatiert und anschließend hinter den letzten auf der Spur vorhandenen Satz der neue Satz geschrieben. Gibt es jedoch auf der Spur nicht mehr genügend Platz, so wird die Bedingung „kein Platz gefunden" signalisiert; der Satz wird also nicht aufgezeichnet. Im EA-Gebiet, auf das durch IOAREA1 verwiesen wird, muß das achtstellige Kennzeichnungsfeld, das Schlüsselfeld (falls Schlüssel verwendet werden) und schließlich das Datenfeld stehen. Eine Prüfung auf falsche Satzlänge ist hier nicht möglich; BLKSIZE muß daher auf eine so große Länge verweisen, daß auch noch der zu schreibende Satz maximaler Länge untergebracht werden kann.

3) WRITE dn,AFTER,EOF
   Im Prinzip lauten die gleichen Operationen wie bei dem unter 2) beschriebenem Makro ab, nur daß hinter dem letzten auf der spezifizierten Spur stehenden Satz ein Dateiendesatz (EOF-Record) aufgezeichnet wird. Die Länge des Datenfeldes eines solchen Satzes ist gleich Null. Wenn ein EOF-Satz gelesen wird, wird die entsprechende Bitstelle der Fehlerbytes (ERRBYTE-Eintragung) auf 1 gesetzt.

Wird der Operand

SRCHM=YES

angegeben und erfolgt der Zugriff über den Schlüssel, so wird das Suchen von der Bezugsspur bis zum Zylinderende fortgesetzt, zu welcher Zeit die Bedingung „Zylinderende" (EOC) ins Mitteilungsbyte ERRBYTE gesetzt wird. Hierbei handelt es sich um eine andere Bedingung als die Bedingung „kein Satz gefunden" (NRF), wenn der Satz nicht ermittelt werden konnte.

Wird der Operand IDLOC benutzt, so verweist er auf ein vierstelliges (RELTYPE=HEX), zehnstelliges (RELTYPE=DEC) oder fünfstelliges Feld, das eine Wertzuweisung vom Makro WAITF erfährt. Werden bei diesem Makro AFTER oder RZERO benutzt, so wird dieses Feld auf einen Scheinwert gesetzt (nur sedezimale Ziffern F oder nur dezimale Ziffern 9 mit Zonen). Bei einem READ oder WRITE mit dem Operanden ID wird diesem Feld ein Wert zugewiesen, der auf den nächsten Satz in der Datei verweist, wenn SRCHM angegeben ist, anderenfalls auf den gleichen Satz. Bei Benutzung von IDLOC müssen alle Plattenbereiche mit Ausnahme des letzten an Zylindergrenzen enden.

In der Abb. 21.16. wird der Inhalt des Mitteilungsbytes ERRBYTE gezeigt.

Byte	*Bitstelle*	*Bedeutung*
0	1	Satz mit falscher Satzlänge
	2	Nichtdatengebundener Übertragungsfehler: Wenn ERREXT angegeben ist und diese Bitstelle steht auf 0, so wurden Daten gelesen oder geschrieben
	4	Kein Platz gefunden: WRITE mit AFTER fand nicht genügend Platz, um den Satz auf die vorgegebene Spur zu schreiben
	7	Bezugnahme auf einen Platz außerhalb des Plattenbereiches: das bei IDLOC angegebene Feld wird auf einen Scheinwert gesetzt
1	0	Datenfehler im Kennzeichnungsfeld: Fehler ist nicht behebbar
	1	Spurenüberlauf: Die Anzahl der auf die Spur aufzuzeichnenden Bytes überschreitet deren Aufnahmevermögen
	2	Zylinderende (EOC): Es wurde der Operand SRCHM angegeben und dennoch der Satz nicht gefunden
	3	Datenfehler im Schlüsselfeld oder im Datenfeld: Fehler ist nicht behebbar
	4	Kein Satz gefunden (NRF): Satz existiert nicht oder Auftreten eines Maschinenfehlers
	5	Dateiende (EOF): Datenträgerende erreicht oder Lesen eines Dateiendesatzes
	6	Datenträgerende (EOV): Dabei wird auch EOC gesetzt

Abb. 21.16. Inhalt des Mitteilungsbytes ERRBYTE

Wenn XTNTXIT angegeben ist, so wird die unter diesem Operanden genannte Routine zur Verarbeitung von Steuerkarten mit Bereichsinformationen herangezogen. Nach der Prüfung der EXTENT-Karten durch das IOCS wird ein 14stelliges Feld erzeugt, das die in Abb. 21.17. aufgeführten Informationen enthält.

Stellen (Bytes)	*Inhalt*	
0	Typenschlüssel	
1	Laufende Bereichsnummer	
2 bis 5	Untere Grenze: ZZKK (Hylinder, Kopf)	
6 bis 9	Obere Grenze: ZZKK	
10 und 11	Symbolische Einheit	
12	Alter Behälter	nur bei IBM 2321
13	Neuer Behälter	

Abb. 21.17. Inhalt des 14stelligen Feldes nach Prüfung des EXTENT-Steuerstatements

Die symbolischen Einheiten werden dabei wie folgt verschlüsselt:

a) Das erste Byte enthält 01 (sedezimal); ihm folgt im zweiten Byte als Dualzahl die unter SYSnnn genannte Nummer.
b) Für logische Systemeinheiten gilt

ØØØØ	für	SYSRDR
ØØØ1	für	SYSIPT
ØØØ2	für	SYSPCH
ØØØ3	für	SYSLST
ØØØ4	für	SYSLOG
ØØØ5	für	SYSLNK
ØØØ6	für	SYSVIS
ØØØ7	für	SYSCAT
ØØØ8	für	SYSCLB
ØØØ9	für	SYSRLB
ØØØA	für	SYSSLB
ØØØB	für	REC

Während der Eröffnung der einzelnen Bereiche stellt die zu OPEN gehörende Routine die Adresse des 14stelligen Feldes ins Register 1 und geht zur Benutzerroutine über. Das Makro LBRET muß diese abschließen, wenn man wieder ins IOCS zurückkehren will.

21.13. Die Dateidefinition DTFIS

In der Abb. 21.18. sind die Eintragungen dargestellt, die man bei der Definition einer logisch fortlaufend (indexsequentiell) organisierten Datei verwenden kann.

Durch die Eintragung IOROUT wird die Art der Verarbeitung festgelegt. LOAD bedeutet, daß die Datei erstellt werden soll. LOAD bedeutet aber auch, daß eine bereits bestehende Datei erweitert werden soll, und zwar so, daß neue Sätze hinter dem ursprünglichen Dateiende hinzugefügt werden sollen. Wenn die Erstellung oder die Erweiterung einer indexsequentiell organisierten Datei vorgesehen ist, so müssen die Sätze in aufsteigender Reihenfolge vorliegen. Es kommt zum Auftreten eines Reihenfolgefehlers, wenn diese Voraussetzung verletzt wird. Beim Vorhandensein von zwei oder mehr Sätzen mit dem gleichen Schlüssel wird ein Duplikatsatzfehler angezeigt.

Durch die Angabe IOROUT=ADD wird festgelegt, daß neue Sätze zur Datei hinzugefügt werden; diese Sätze können dann an beliebigen Stellen (blindlings) in die Datei hineingeschoben werden, d.h. sie brauchen nicht in aufsteigender Reihenfolge vorliegen. IOROUT=RETRVE bedeutet, daß Sätze von der Datei gelesen und möglicherweise auch aufdatiert werden sollen; neue Sätze sollen jedoch nicht hinzugefügt werden. Die Angabe von ADDRTR führt zur Ausübung der beiden eben beschriebenen Funktionen, also der ADD- und der RETRVE-Funktion. Wenn man nur ADD oder RETRVE angibt, führt das zu dem Vorteil, daß eine IOCS-Routine geringeren Umfangs verwendet wird, wodurch man Hauptspeicherplatz einsparen kann. Da der LOAD-Vorgang eine fortlaufende Verarbeitung und der ADD-Vorgang Hinzufügungen an gestreuten Stellen einschließt, wird die TYPEFLE-Angabe nur für RETRVE und ADDRTR benötigt. Wenn SEQNTL spezifiziert wird, so müssen die Makros READ und WRITE benutzt werden. Die Datei erweist sich dann in gewisser Hinsicht wie eine gestreut organisierte Datei. Bei der Angabe von

Operand	*Bedeutung*	*Bemerkung*
CYLOFL=n	Zylinderüberlaufbereich	Für n ist die Anzahl der Spuren pro Zylinder des Überlaufbereiches einzusetzen
DEVICE={2311 / 2314 / 2321 / 333Ø / 334Ø}	Festlegung des Gerätetyps	Für IBM 2311 nicht erforderlich
DSKXTNT=n	Anzahl der Dateibereiche	Für jede Datei erforderlich
ERREXT=YES	Verwendung des Makros ERET	--
HINDEX={2311 / 2314 / 2321 / 333Ø / 334Ø}	Höchste Stufe des Inhaltsverzeichnisses auf einem Gerät des genannten Typs	Für IBM 2311 nicht erforderlich
INDAREA=name	Hauptspeicherbereich für das Zylinderinhaltsverzeichnis	INDSIZE muß ebenfalls angegeben werden
INDSIZE=n	Zylinderinhaltsverzeichnis im Hauptspeicher (Gebietslänge)	n Länge des Zylinderinhaltsverzeichnisses
INDSKIP=YES	Ermöglicht das Übergehen der ersten Sätze einer Datei	INDAREA und INDSIZE müssen ebenfalls angegeben sein
IOAREAL=name	EA-Gebiet beim Laden	Bei IOROUT darf nicht RTRVE angegeben sein
IOAREAR=name	EA-Gebiet beim direkten Zugriff	Bei TYPEFLE darf nicht SEQNTL angegeben sein
IOAREAS=name	EA-Gebiet beim fortlaufenden Zugriff	Bei TYPEFLE darf nicht RANDOM angegeben sein
IOAREA2=name	2. EA-Gebiet	Nur möglich beim sequentiellen Laden und beim sequentiellen Zugriff
IOREG=(r)	Verarbeitung geblockter Sätze im EA-Gebiet	Unter r ist eines der Register 2 bis 12 anzugeben
IOROUT={LOAD / ADD / RETRVE / ADDRTR}	LOAD: Aufbau oder Erweiterung einer Datei ADD: Hinzufügen neuer Sätze zu einer existierenden Datei RETRVE: Wiederauffinden von Sätzen (Aufdatierung erlaubt) ADDRTR: Wiederauffinden und Hinzufügen von Sätzen	Für jede Datei erforderlich

Abb. 21.18. Mögliche Eintragungen bei der Dateidefinition DTFIS (1. Teil)

Operand	*Bedeutung*	*Bemerkung*
IOSIZE=n	Länge des EA-Gebietes (unter IOAREAL) beim Laden	EA-Gebiet größer als die minimal erforderliche Länge
KEYARG=name	Fortlaufendes durch den angegebenen Schlüssel eingeleitetes Lesen; direktes Lesen	--
KEYLEN=n	Schlüssellänge n	Für jede Datei erforderlich
KEYLOC=n	Schlüsselposition bei geblockten Sätzen	n erste Schlüsselstelle innerhalb eines Satzes relativ zu 1
MSTIND=YES	Verwendung eines Hauptinhaltsverzeichnisses (Hauptindex)	--
NRECDS=n	Blockungsfaktor der Sätze	Für jede Datei erforderlich; bei n = 1 liegen ungeblockte Sätze vor
RECFORM={FIXUNB / FIXBLK}	Satzformat	Angabe für jede Datei erforderlich
RECSIZE=n	Satzlänge	Für jede Datei erforderlich
TYPEFLE={RANDOM / SEQNTL / RANSEQ}	Verarbeitungsform	RANDOM | Direkter Zugriff SEQNTL | Fortlaufender Zugriff RANSEQ | Direkter und fortlaufender Zugriff
VERIFY=YES	Schreibprüfung	Nur bei Plattenspeichern erforderlich
WORKL=name	Verwendung eines Arbeitsgebietes	Voraussetzung ist die Angabe von IOAREAL
WORKR=name	Verwendung eines Arbeitsgebietes (direkter Zugriff)	Voraussetzung ist die Angabe von IOAREAR und die Verarbeitung im Arbeitsgebiet; der Parameter IOREG bleibt unbeachtet
WORKS=YES	Verwendung eines Arbeitsgebietes (Bezugnahme in GET- oder PUT-Makros)	Voraussetzung ist die Angabe von IOAREAS und die Verarbeitung im Arbeitsgebiet; der Parameter IOREG bleibt unbeachtet

Abb. 21.18. Mögliche Eintragungen bei der Dateidefinition DTFIS (2. Teil)

RANSEQ können GET- und READ-Makros eingesetzt werden; das Schreiben und Aufdatieren von Sätzen ist in diesem Fall nicht erlaubt.

Die Umfänge der EA-Gebiete und der Arbeitsgebiete, die bei der indexsequentiellen Verarbeitung benutzt werden müssen, sind in der Abb. 21.19. festgehalten.

	Eingabe/Ausgabe-Gebiet		*Arbeitsgebiete*	
	FIXUNB	FIXBLK	FIXUNB	FIXBLK
Laden (Erstellen) einer Datei	8 + KL + RL	8 + KL + RL * NRECDS	KL + RL	RL
Hinzufügen neuer Sätze	8 + KL + 10 + RL	8 + KL + 10 + RL 8 + KL + RL * NRECDS	KL + RL	RL
Direktes (wahlweises) Wiederauffinden (Lesen)	10 + RL	10 + RL RL * NRECDS	RL	RL
Fortlaufendes (sequentielles) Wiederauffinden (Lesen)	10 + KL + RL	10 + KL + RL 10 + KL + RL * NRECDS	KL + RL	RL

Anmerkungen: 1) Es bedeuten: | RL Satzlänge
| KL Schlüssellänge

2) Ergeben sich bei Anwendung der Formeln verschiedene Zahlenwerte, so ist stets der größere zu benutzen

Abb. 21.19. Umfänge der EA-Gebiete und der Arbeitsgebiete bei indexsequentiell organisierten Dateien

Besprechen wir zunächst einmal die Vorgänge beim Laden einer Datei. Zu Beginn schicken wir das Makro

```
SETFL dateiname
```

hinaus. Vor diesem Makro muß das Makro OPEN erscheinen. Jeder hinzuzufügende Satz muß einen Schlüssel in aufsteigender Reihenfolge aufweisen. Der im WORKL-Gebiet stehende Satz kommt zur Datei hinzu, wenn das Makro

```
WRITE dateiname,NEWKEY
```

ausgeführt wird. Sind die Sätze ungeblockt, so muß der Schlüssel den Daten im WORKL-Gebiet vorausgehen. Durch IOSIZE werden die Umfänge von IOAREL und IOAREAZ festgelegt. Dieser Wert darf nicht die Spurkapazität übersteigen. Ist er größer als der in Abb. 21.19. aufgeführte Wert, so kann die Leistungsfähigkeit erhöht werden.

Nach dem Laden des letzten Satzes ist die Ausführung des Makros

```
ENDFL dateiname
```

erforderlich, damit der letzte Block und ein Dateiendesatz (EOF-Satz) geschrieben werden. Danach kann die Datei geschlossen werden.

Das Hinzufügen von Sätzen zu einer Datei besorgen die Makros SETFL und ENDFL nicht, jedoch das Makro WRITE mit dem Operanden NEWKEY. Jeder Ausführung eines WRITE-Makros sollte sich die Untersuchung des Bytes mit dem Namen dateiname C anschließen, um nach aufgetretenen Fehlern zu sehen. Die Bedeutung der Bitstellen dieses Bytes kann der Abb. 21.20. entnommen werden.

	IOROUT=LOAD
Bit 0	Auftreten eines Fehlers beim Plattenspeicher (jedoch nicht falsche Satzlänge)
Bit 1	Satz mit falscher Satzlänge
Bit 2	Primärdatenbereich gefüllt bei IOROUT=LOAD, sonst Dateiende (EOF)
Bit 3	Zylinderindexbereich gefüllt
Bit 4	Hauptinhaltsverzeichnis erschöpft
Bit 5	Duplikatsatz (der Satz mit dem angegebenen Schlüssel existiert bereits
Bit 6	Reihenfolgefehler (bei den Schlüsseln der Sätze)
Bit 7	Überlauf des Primärdatenbereiches (nicht genügend Platz für den Dateiendesatz vorhanden)

	IOROUT=LOAD
Bit 0	Auftreten eines Plattenspeicherfehlers (jedoch nicht falsche Satzlänge)
Bit 1	Satz mit falscher Satzlänge
Bit 2	Dateiende (EOF)
Bit 3	Kein Satz gefunden
Bit 4	Angabe einer ungültigen Kennzeichnung (SETL) außerhalb der Dateigrenzen
Bit 5	Duplikatsatz
Bit 6	Überlaufbereich gefüllt (unabhängiger Überlaufbereich oder Zylinderüberlaufbereich gefüllt und es existiert kein unabhängiger Überlaufbereich)
Bit 7	Überlauf (Der gerade angesprochene Satz befindet sich im Überlaufbereich; es liegt in diesem Fall kein Fehler vor)

Abb. 21.20. Bedeutung der Bitstellen des Bytes mit der Bezeichnung dateiname C

Der direkte Zugriff zu einem freigewählten Satz erfordert zuvor, daß das Suchargument nach dem bei KEYARG genannten Feld gebracht wird. Jede Eingabeoperation wird durch die Ausführung des Statements

```
READ  dateiname,KEY
```

Die Vollendung der Leseoperation wird durch die Ausführung des Makros

```
WAITF dateiname
```

abgesichert. Die Bitstellen von dateiname C werden hierbei gesetzt. Ein einmal gelesener Satz kann durch die Ausführung des Statements

```
WRITE dateiname,KEY
```

ersetzt werden; ein WAITF-Makro muß folgen.

Während des Lesens wird das acht Byte lange Feld mit der Bezeichnung dateiname G mit MBBZZKKR (Datenträger, Zelle, Zylinder, Kopf, Satz) gefüllt, d.h. mit der Kennzeichnung, die den gelesenen Satz auszeichnet. Wird dieser Wert nachfolgend konsequent benutzt, so sollten wir uns daran erinnern, daß beim Hinzufügen von Sätzen Kennzeichnungen geändert werden können.

Beim Wiederaufsuchen von Sätzen in einer Datei, deren Definition sich durch

```
IOROUT=ADDRTR
```

auszeichnet, können sich WRITE-Makros mit NEWKEY und WRITE-Makros mit KEY in beliebiger Weise abwechseln.

Die sequentielle Verarbeitung wird durch das Makro

```
SETL  dateiname,start
```

eingeleitet. An Stelle von start ist einer der nachfolgenden Parameter zu codieren:

BOF	Die Verarbeitung soll am Dateianfang beginnen
KEY	Die Verarbeitung soll bei dem Satz beginnen, dessen Schlüssel gleich dem Wert ist, der im Feld gespeichert ist, das unter KEYARG erwähnt ist
GKEY	Die Verarbeitung soll bei dem Satz beginnen, dessen Schlüssel gleich oder größer als der Wert ist, der im Feld gespeichert ist, das unter KEYARG erwähnt ist
idname	Die Verarbeitung soll bei dem Satz beginnen, dessen Kennzeichnung MBBZZKKR auf dem unter idname genannten Feld steht. Für idname kann auch eine Registernotierung (r) geschrieben werden, wobei r eines der Register 2 bis 12 einschließlich ist

Die Sätze werden durch die Ausführung des READ-Makros bereitgestellt. Es kann in zwei Formen verwendet werden:

```
GET   dateiname
GET   dateiname,arbeitsgebietname
```

Welche der beiden Formen anzuziehen ist, hängt davon ab, ob in der Dateidefinition

```
WORKS=YES
```

angegeben ist. In jedem Fall wird jedoch während der Ausführung das Byte mit der Bezeichnung dateiname C wertmäßig versorgt und die Kennzeichnung des Satzes ins Feld mit dem Namen dateiname H plaziert. Der Datenteil des Satzes kann durch die Ausführung des Statements

```
PUT   dateiname
PUT   dateiname,arbeitsgebietname
```

ersetzt werden. Welche Form einzusetzen ist, hängt ebenfalls davon ab, ob in der Dateidefinition

```
WORKS = YES
```

angegeben ist. Der als 2. Operand erscheinende Arbeitsgebietname braucht nicht der gleiche zu sein, der im GET-Makro spezifiziert ist. Wird kein Arbeitsgebietname benötigt, so bestimmt der Inhalt des Registers bei IOREG die Stelle des Satzes. Sind die Sätze ungeblockt, so geht der Schlüssel den Daten im Eingabegebiet bzw. im Arbeitsgebiet voraus.

Die sequentielle Verarbeitung schließt ab, wenn die Programmablaufsteuerung dem Makro

```
ESETL dateiname
```

begegnet.

Bei Spezifikation des Operanden

TYPEFLE=RANSEQ

in der Dateidefinition kann sowohl die direkte als auch die sequentielle Verarbeitung durchgeführt werden, wobei jeder sequentielle Verarbeitungsabschnitt mit SETL beginnen und mit ESETL enden muß. Die direkte Verarbeitung kann nach einem ESETL und vor dem nächsten SETL eingeschleust werden.

Für dateiname in SETFL und ENDFL kann auch eine Registerangabe, und zwar (0) oder (r) geschrieben werden; für dateiname in den anderen Makros besteht die Möglichkeit von (1) oder (r). Arbeitsgebietsnamen können durch (0) oder (r) ausgedrückt werden. Unter r ist jeweils eins der allgemeinen Register 2 bis 12 zu verstehen.

21.14. Die Moduln des IOCS

Wir haben nunmehr acht Dateidefinitionen kennengelernt, durch die im Betriebssystem DOS die Eigenschaften von Dateien festgelegt werden, die vom logischen IOCS angesprochen werden sollen. Jedes DTF-Statement erzeugt für die betreffende Datei eine Tabelle von Konstanten. Eine dieser Konstanten ist eine Adreßkonstante vom V-Typ: sie bezieht sich auf den Namen einer standardmäßigen IOCS-Routine. Es stehen mehrere Hundert dieser IOCS-Routinen zur Verfügung. Im allgemeinen wird jede Installation in ihrer Bibliothek für verschiebliche Programme (Bibliothek für Objektmoduln) diejenigen Routinen katalogisieren, die am häufigsten benutzt werden. Wenn eine solche Katalogisierung vorliegt, wird zur Verbindungs- und Aufbereitungszeit (Programmablaufzeit des Linkage Editors) durch die selbsttätige Verbindungseinrichtung des DOS die Bibliothek der Objektmoduln nach der Routine durchsucht, deren Name in der Adreßkonstanten des V-Typs steht. Diese Routine wird automatisch dem aufzubereitenden Programm einverleibt. Dieses automatische Durchsuchen wird immer dann eingeleitet, wenn eine externe Bezugnahme innerhalb des aufzubereitenden Moduls nicht aufgelöst werden kann.

Sollte es passieren, daß die angeforderte Routine nicht in der Bibliothek enthalten ist, so wird der Binder eine Fehlermeldung ausgeben.

Das Betriebssystem DOS ermöglicht es dem Benutzer, daß die passende IOCS-Routine entweder als Teil seines Programmes oder vollständig getrennt von irgendeinem Programm generiert wird. Wenn die Routine als Teil seines Programmes erzeugt wird, so ist ein Durchsuchen der Bibliothek nicht erforderlich. Wenn die Routine dagegen getrennt generiert wird, so kann der Objektmodul gestanzt werden und in die Bibliothek für Objektmoduln für eine zukünftige Verwendung hineingestellt werden. Bei Benutzung der Dateidefinition DTFCN wird die entsprechende IOCS-Routine stets als Teil des Codes erzeugt, der sich aus der Dateidefinition ergibt. Aus allen anderen Dateidefinitionen werden nur die Konstanten und die Adreßkonstante des V-Typs für die einzelnen Dateien generiert. Der Programmierer muß selbst dafür Sorge tragen, daß sein Programm mit der entsprechenden IOCS-Routine ergänzt wird. Er kann diese Routine entweder zur Verbindungs- und Aufbereitungszeit über Karten hinzufügen, denen eine INCLUDE-Steuerkarte vorausgeht, oder er kann sie aus der Bibliothek für Objektmoduln holen. Es bietet sich ihm auch noch eine dritte Möglichkeit an, wenn er in sein Programm ein xxMOD-Makro aufnimmt. Es gibt die folgenden sieben verschiedenen Makros: CDMOD, PRMOD, MTMOD, SDMOD, DAMOD, DIMOD und ISMOD. Aus ihnen wird vom Assemblierer eine IOCS-Routine erzeugt, die der entsprechenden Dateidefinition gerecht wird. Diesen Routinen werden Standardnamen zugeordnet. Wir wollen die einzelnen Makros der Reihe nach betrachten.

1) CDMOD

Für dieses Makro gibt es die folgenden Operanden:

RECFORM= { FIXUNB / VARUNB / UNDEF }

CTLCHR= { YES / ASA }

CONTROL=YES

TYPEFLE= { INPUT / OUTPUT / COMBND }

WORKA=YES

IOAREA2=YES

CRDERR=RETRY

DEVICE=g

FUNC=x

Für g und x können die gleichen Angaben, die bei den entsprechenden Operanden des Makros DTFCD (siehe Abb. 21.4.) vorliegen können, gemacht werden.

Die Auswahl der geeigneten Eintragungen bewirkt die Generierung eines Moduls, das die gewünschte DTFCD-Definition unterstützt. Dieses Modul besitzt den Namen

IJCabcde

An Stelle der Kleinbuchstaben treten in Abhängigkeit von den spezifizierten Eintragungen die folgenden Großbuchstaben:

für a: In Abhängigkeit von RECFORM erscheint F, V oder U

für b: Es erscheint Y, wenn CTLCHR=YES; A, wenn CTLCHR=ASA; C, wenn CONTROL=YES und Z, wenn nichts spezifiziert ist.

für c: I, O oder C in Abhängigkeit von der TYPEFLE-Angabe.

für d: W, wenn WORKA; I, wenn IOAREA2; B, wenn beide und Z, wenn keine Spezifizierung vorliegt.

für e: 0 bei der IBM 2540, 1 bei der IBM 2596, 2 bei der IBM 2520, 3 bei der IBM 2501, 4 bei der IBM 2540 und Angabe von CRDERR, 5 bei der IBM 2520 und Angabe von CRDERR, 6 bei der IBM 3504 und der IBM 3505, 7 bei der IBM 3525 und FUNC=R oder FUNC=P usw.

Somit unterstützt also das Modul IJCFZIW3 eine DTFD-Definition für eine Datei, die einer IBM 2501 zugeordnet ist und von der ungeblockte Sätze fester Länge in ein Arbeitsgebiet eingelesen werden.

2) PRMOD

Für diese Definition gibt es die folgenden Operanden:

REFORM={FIXUNB / VARUNB / UNDEF}

CTLCHR={YES / ASA}

CONTROL=YES

PRINTOV=YES

ERROPT=YES

FUNC=x

IOAREA2=YES

WORKA=YES

Für x können die gleichen Angaben erscheinen, die auch bei der Definition DTFPR zulässig sind (siehe Abb. 21.6.). Die Namen der generierten Moduln heißen dann

IJDabcde

Die Kleinbuchstaben werden wie folgt durch Großbuchstaben ersetzt:

für a: F, V oder U (genau so wie bei CDMOD)
für b: Y, A, C oder Z (genau so wie bei CDMOD); T für IBM 3525 mit 2 Druckzeilen, U für die IBM 2560 und V für die IBM 5425
für c: P bei PRINTOV; Z, wenn keine PRINTOV-Angabe vorliegt
für d: I bei Angabe von IOAREA2; Z, wenn diese Angabe fehlt.
für e: W bei Angabe von WORKA; Z, wenn diese Angabe fehlt.

(Wie üblich werden die unterstrichenen Angaben angenommen, wenn die betreffende Eintragung weggelassen wird; das gilt für alle Modulbesprechungen). Gelegentlich kann die Tatsache von Wichtigkeit sein, daß einige Moduln in anderen Moduln enthalten sind. So ist z. B. IJDFAZIW Bestandteil des Moduls IJDFAPIW. Die Verwendung eines größeren Moduls als des notwendigen ist erlaubt; es wächst nur der Hauptspeicherbedarf an, das Objektprogramm bleibt genau so wirkungsvoll. Die Benutzung der umfassenderen Moduln wird oft geplant, weil man damit das Schreiben einer anderen xxMOD-Definition vermeiden kann. Zu dieser Maßnahme greift man zweckmäßigerweise auch dann, wenn zwei oder mehr Dateien an der gleichen IOCS-Routine teilhaben können.

3) MTMOD

Es gibt folgende Operanden:

RECFORM={FIXUNB / FIXBLK / VARUNB / VARBLK / UNDEF}

TYPEFLE={INPUT / OUTPUT / WORK}

READ={FORWARD / BACK}

CKPTREC=YES
WORKA=YES

Die Namen der Moduln sind

IJFabcdz

An Stelle der Kleinbuchstaben treten Großbuchstaben wie folgt:

für a: F, V, S oder U (das Blocken bleibt unberücksichtigt)
für b: B bei READ=BACK, sonst Z
für c: C bei Angabe von CKPTREC, sonst Z. (Durch CKPTREC werden Prüfpunktsätze ermöglicht; diese haben wir noch nicht behandelt).
für d: W bei Angabe von WORKA=YES, Z beim Fehlen dieser Angabe.

Für Arbeitsdateien auf Bändern kann man, abgesehen von der TYPEFLE-Eintragung, noch folgendes spezifizieren:

ERROPT=YES

NOTEPNT={YES / POINTS}

Der Modulname ist in diesem Fall

IJFWbcZZ

An Stelle von b tritt E, wenn ERROPT spezifiziert ist, sonst Z. Der Buchstabe c wird durch N ersetzt, wenn NOTEPNT=YES angegeben ist, durch S bei der Angabe von NOTEPNT=POINTS und durch Z, wenn keine dieser Angaben vorliegt.

4) SDMOD

Wegen der Länge der Moduln, die für Einheiten mit direkter Zugriffsmöglichkeit generiert werden, gibt es zehn verschiedene Makros, mit deren Hilfe die SD-Moduln erzeugt werden. Durch neun von ihnen werden IOCS-Routinen generiert, die für gewöhnliche SEDZ-Dateien gelten. Diese Makros besitzen die Form

SDMODrt

für r sind die folgenden Buchstaben einzusetzen: F bei Dateien mit Sätzen fester Länge, V bei Dateien mit Sätzen variabler Länge, U bei Dateien mit Sätzen undefinierter Länge und S bei Dateien mit Spannsätzen. An Stelle von t sind die folgenden Buchstaben einzusetzen: I bei Eingabedateien, O bei Ausgabedateien und U bei Dateien, die der Aufdatierung unterliegen sollen. Das zehnte Makro ist

SDMODW

Durch dieses Makro wird die für Arbeitsdateien benötigte IOCS-Routine erzeugt. Da diese Makros aus mehr als fünf Zeichen bestehen, passen ihre Codes nicht in das Operationsschlüsselfeld der Assemblierer-Programmschemen. Der Makroschlüssel beginnt zwar auf derselben Spalte wie die Operationsschlüssel der Instruktionen, doch wird er über das Operationsfeld hinaus fortgesetzt. Der Makro-Operationsschlüssel muß jedoch durch mindestens eine Leerstelle vom ersten Operanden getrennt werden. Bei erforderlichen Fortsetzungszeilen wird ein Makrostatement auf der gleichen Spalte wie jedes andere Statement fortgesetzt, d.h. in Spalte 16.

Bei SDMOD-Definitionen sind folgende Eintragungen möglich:

```
CONTROL=YES
ERROPT=YES
ERREXT=YES
TRUNCS=YES
NOTEPNT={POINTRW}
        {YES    }
FEOVD=YES
UPDATE=YES
```

CONTROL und ERROPT können bei allen SDMOD-Definitionen angewendet werden. TRUNCS kann nur für Dateien mit Sätzen fester Länge angegeben werden. Die beiden letzten Operanden (NOTEPNT und UPDATE) können nur bei Arbeitsdateien aufgeführt werden. Mit Ausnahme der Arbeitsdateien werden die Modulnamen wie folgt gebildet:

IJGabcde

An Stelle der Kleinbuchstaben treten die nachstehenden Großbuchstaben:

für a: F, V, S oder U vom Makronamen
für b: I, O oder U vom Makronamen
für c: E, wenn ERROPT spezifiziert ist, C wenn daneben noch ERREXT angegeben ist, sonst Z
für d: T, wenn TRUNCS spezifiziert ist, W bei Angabe von FEOVD, M bei Angabe von beiden Operanden, sonst Z
für e: C, wenn CONTROL spezifiziert ist, sonst Z

Bei Arbeitsdateien lauten die Modulnamen

IJGWcfeg

Die Kleinbuchstaben c und e werden genau so ersetzt wie bei den regulären Dateien; an Stelle von f treten die folgenden Buchstaben:

R, wenn NOTEPNT=POINTRW spezifiziert ist,
N, wenn NOTEPNT=YES spezifiziert ist und
Z, wenn nichts spezifiziert ist.

An Stelle von g erscheint U, wenn UPDATE=YES vorliegt, sonst Z.

5) DAMOD

Es gibt hierbei die folgenden Operanden:

RECFORM=s
FEOVD=YES
ERREXT=YES
RELTRK=YES
AFTER=YES
IDLOC=YES

Die Namen der entsprechenden Moduln lauten:

IJIabcdZ

Die Kleinbuchstaben werden wie folgt ersetzt:

für a: F, V, S oder B (je nach der definierten Satzart s, B bei FIXUNB und UNDEF)
für b: A, wenn AFTER=YES spezifiziert ist, sonst Z
für c: I, wenn IDLOC=YES angegeben ist, R, wenn FEOVD=YES vorliegt, E bei Angabe von beiden Operanden, sonst Z
für d: P bei Angabe von ERREXT=YES, R bei RELTRK=YES, H bei Angabe von beiden, sonst Z

Für s kann in der Definition des Moduls dasselbe geschrieben werden, was an Angaben im Makrostatement DTFDA zulässig ist (siehe Abb. 21.15.).

6) DIMOD

Der Modulname heißt IJJFCcdD. An Stelle der Kleinbuchstaben treten bei der Generierung die folgenden Großbuchstaben:

für c: I, wenn TYPEFLE=INPUT vorliegt, sonst B (bei B werden sowohl Eingabe- als auch Ausgabedateien verarbeitet).
für d: I bei Angabe von IOAREAZ=YES, sonst Z.

7) ISMOD

Es gibt die folgenden Operanden:

IOROUT= { LOAD / ADD / RETRVE / ADDRTR }
ERREXT=YES
IOSIZE=n
RECFORM= { FIXUNB / FIXBLK / BOTH }
TYPEFLE= { RANDOM / SEQNTL / RANSEQ }
CORINDX=YES
CORDATA=YES
IOAREA2=YES

Die Angabe von n bei IOSIZE hat die gleiche Bedeutung wie beim Makro DTFIS (siehe Abb. 21.18.). Die Modulnamen lauten

IJHabcde

Die Kleinbuchstaben werden wie folgt ersetzt:

für a: U bei Angabe von FIXUNB und wenn b gleich I oder A ist,
B bei Angabe von FIXBLK und wenn b gleich I oder A ist,
A bei Angabe von BOTH und wenn b gleich I oder A ist,
Z, wenn LOAD oder RETRVE spezifiziert ist (dieser Buchstabe schließt BOTH ein)

für b: L bei Angabe von LOAD, I bei Angabe von ADD, R bei Angabe von RETRVE und A bei Angabe von ADDRTR (bei IOROUT)

für c: R bei Angabe von RANDOM, S bei Angabe von SEQNTL, G bei Angabe von IOAREA2, B bei Angabe von RANSEQ und Z, wenn die Angabe des Operanden TYPEFLE fehlt (TYPEFLE muß jedoch aufgeführt sein, wenn für IOROUT die Angaben RETRVE oder ADDRTR vorliegen)

für d: C, wenn CORINDX=YES angegeben ist, sonst Z

für e: G bei Angabe von IOSIZE und ERREXT, P bei alleiniger Angabe von IOSIZE, T bei alleiniger Angabe von ERREXT, sonst Z.

Zu jedem der eben besprochenen xxMOD-Makrostatements kann die Eintragung SEPASMB=YES hinzugefügt werden. Durch sie wird festgelegt, daß die Dateidefinitionen vom Assembler nicht im gleichen Lauf mit übersetzt werden. Aus diesem Grunde müssen deshalb die Routinen ENTRY-Statements enthalten; dadurch kann der Modul von anderen Programmen aufgerufen werden.

Mit Ausnahme der Dateidefinition DTFCN ist für jede andere Dateidefinition auch die Eintragung

```
MODNAME=name
```

erlaubt. Hierbei ist unter name der Name des Moduls anzugeben, der in das Programm eingeschlossen werden soll. Wenn mehr als eine Datei den gleichen Modul benutzen kann, dann kann der Programmierer diesen Modul für alle geeigneten DTF-Definitionen zusammen spezifizieren, d.h. er braucht nur einmal ins Programm eingefügt zu werden. Selbstverständlich muß er dabei sorgfältig darauf achten, daß der generierte Modul auch alle Operationen unterstützt, die er beabsichtigt, ausführen zu lassen.

Auf der anderen Seite kann zu allen Moduldefinitionen der Operand

```
SEPASMB=YES
```

hinzugefügt werden. Durch ihn wird festgelegt, daß die generierte Routine auch ENTRY-Statements einschließt. Die Verwendung dieses Operanden ist unumgänglich notwendig, wenn der Modul separat von den Dateidefinitionen übersetzt wird, was wohl in der Regel gemacht werden dürfte.

Um die Verwendung der Moduln zu veranschaulichen, wollen wir die Beispiele des Abschnitts 21.8. betrachten. Wenn wir bei unserem ersten Beispielprogramm (Abbildung 21.9.) zusätzlich die Statements

```
CDMOD  WORKA=YES,DEVICE=254Ø
MTMOD  WORKA=YES
```

codiert hätten, so würden vom Assemblierer automatisch in unser Programm die beiden IOCS-Moduln

IJCZIWO und IJFFZZWZ

eingefügt worden sein.

Beim Programmbeispiel 2 vom Abschnitt 21.8. (siehe auch Abb. 21.10.) bleibt der Operand von MTMOD bestehen, doch für CDMOD hätten wir schreiben müssen:

```
CDMOD DEVICE=254Ø
```

Auf diese Weise wäre uns der Modul IJCFZIZO generiert worden. Beim Programmbeispiel 3 (siehe auch Abb. 21.11.) hätten die Operanden von CDMOD die gleichen wie beim ersten Beispiel sein müssen; für MTMOD brauchten wir hier jedoch keine Operanden anzugeben. Damit würde aber der Modul IJFFZZZZ an Stelle von IJFFZZWZ generiert.

Beim vierten Beispielprogramm (siehe auch Abb. 21.12.) könnten wir zusätzlich

```
CDMOD DEVICE=254Ø,IOAREA2=YES
MTMOD
```

codieren. Es würden dann die Moduln IJCZII0 und IJFFZZZZ erzeugt.

21.15. Das physische IOCS

Vier Makrostatements stehen für die physische EA zur Verfügung. Ein Befehlssteuerblock (Command Control Block, abgekürzt CCB) wird erstellt, wenn man das Makro CCB verwendet, das die nachstehende Form besitzt:

```
ccbname  CCB   SYSnnn,kanalbefehlswortliste,X'nnnn',prüfadr
```

Die Benutzung der letzten beiden Operanden ist freigestellt. Durch kanalbefehlswortliste wird die symbolische Adresse des ersten CCW des Kanalprogrammes festgehalten. Der erste Operand bezieht sich auf eine beliebige verfügbare logische EA-Einheit des Systemes. Durch das CCB-Makro wird ein 16- oder 24-stelliger Block erzeugt, der so aufgebaut ist, wie es die Abb. 21.21. zeigt.

Wird der dritte Operand weggelassen, so werden die Kennzeichenbytes auf X'ØØØØ' gesetzt; im anderen Fall wird der dritte Operand dazu benutzt, um die Kennzeichenbytes auf einen Anfangswert zu setzen.

Die Ausführung eines Kanalprogrammes wird durch das Makro

```
EXCP  ccbname [,REAL]
```

angestoßen. Der erste Operand nennt den Namen des Befehlssteuerblockes (ccbname). Er darf auch in Form einer Registernotierung (1) oder (r) angegeben werden. Dabei müssen dann das Register 1 bzw. 2 bis einschließlich 12 die Adresse des Kanalsteuerblockes enthalten. Wird der Operand REAL niedergeschrieben, dann nimmt das System an, daß die Adressen in den Kanalbefehlswörter und die Kanalbefehlswortadresse im Kanalstatuswort in reale Adresse übersetzt wurden. Ist REAL nicht codiert, so werden virtuelle Adressen angenommen und der Supervisor führt sie in reale über.

Das Makro

```
WAIT  ccbname
```

bewirkt, daß der Computer die weitere Programmausführung so lange aufschiebt bis die EA-Operation beendet ist, auf die durch ccbname bezug genommen wurde. Der Operand kann ebenfalls in Form einer Registernotierung (1) oder (r) geschrieben werden.

Bytes	Inhalt
0 und 1	Übrigbleibender Zählerwert. Das IOCS stellt den Zählerwert der zuletzt abgelaufenen Kanaloperation in diese beiden Bytes. Dadurch wird die Prüfung der Satzlänge wesentlich erleichtert.
2 und 3	Kennzeichenbytes (Flag Bytes)
4 und 5	Statusbytes des Kanalstatuswortes (CSW) zur Zeit der Kanalendebedingung.
6 und 7	Symbolische Einheit, aus der SYSnnn-Spezifikation ermittelt.
8	00 (sed)
9 bis 11	Adresse des ersten Kanalbefehlswortes (CCW)
12	00 (sed)
13 bis 15	Adresse des CCW, die beim Kanalende aus dem CSW weggespeichert wurde.
16 bis 23	Prüfkanalbefehlswort zur Rückgabe von Prüfinformationen ans Programm. Dieses CCW wird nur dann generiert, wenn prüfadr in die Definition einbezogen ist. Die prüfadr zugeordnete Länge in der Symboltabelle wird als Zähler für das CCW benutzt.

Abb. 21.21. Befehlssteuerblock beim physischen IOCS

Normalerweise sind diese Makros die einzigen, die bei der physischen EA benutzt werden. Besitzt jedoch ein Plattenspeicher oder ein Magnetband Standardkennsätze, dann kann der Benutzer vom IOCS die Behandlung der Kennsätze anfordern, wozu er sich der Makros OPEN, LBRET, CLOSE, FEOV und SEOV bedient (die letzteren beiden sind natürlich nur bei Magnetbändern erlaubt). Um diese Makros einsetzen zu können, muß er eine Dateidefinition niederschreiben. Hierzu ist das Makro

```
name  DTFPH  operand [,operand] ···
```

zu verwenden.

Es gibt folgende Operanden:

```
CCWADDR=name
DEVICE={TAPE|2311|2314|2321|333Ø|334Ø}
DEVADDR=SYSnnn
HDRINFO=YES
LABADDR=name

MOUNTED={ALL|SINGLE}
TYPEFLE={INPUT|OUTPUT}
XTNTXIT=name
```

Durch das Makro DTFPH wird eine Tabelle aufgebaut. In den Bytes 0 bis 15 dieser Tabelle steht der bereits bekannte Befehlssteuerblock (Command Control Block, abgekürzt CCB). Die Bytes 54 bis 57 enthalten ZZKK (Zylinder, Kopf) für die obere Begrenzung des augenblicklich bearbeiteten Plattenspeicherbereiches. In gleicher Form steht in den Bytes 60 bis 63 die untere Begrenzung. Wenn der CCWADDR-Operand angegeben ist, enthalten die Bytes 8 bis 11 die Adresse des ersten Kanalbefehlswortes (CCW), d.h. die symbolische Adresse. Der Name dieses Wortes muß bei CCWADDR=name angegeben sein. Wenn der CCWADDR-Operand weggelassen wird, dann enthalten die Bytes 8 bis 11 die um 8 vergrößerte Adresse des ersten Bytes der DTFPH-Tabelle.

Für die einzelnen Operanden der Dateidefinition halten wir folgende Bemerkungen für angebracht:

a) CCWADDR
Bei Einschluß dieses Operanden wird die angegebene Adresse für den Operanden kanalbefehlsliste des Befehlssteuerblockes CCB benutzt.

b) DEVADDR
Die Angabe dieses Operanden ist notwendig, es sei denn, daß ein auf einem Plattenspeicher gelegene Datei eine Systemadresse durch die erste EXTENT-Steuerkarte zugewiesen erhält.

c) DEVICE
Mit Ausnahme von Magnetbändern ist die Angabe dieses Operanden notwendig.

d) LABADDR
Dieser Operand ist notwendig, wenn es gilt, Benutzervorsätze und Bandnachsätze zu verarbeiten.

e) MOUNTED
Dieser Operand ist für Dateien erforderlich, die SEDZ zugewiesen sind. SINGLE besagt, daß bei der Eröffnung der Datei nur der erste Bereich verfügbar ist. Für jeden zusätzlichen Bereich muß ein weiteres OPEN-Statement vorliegen. ALL dagegen besagt, daß nach dem Eröffnen alle Plattenbereiche verfügbar sind.

f) XTNTXIT
Dieser Operand ist erforderlich, wenn der Benutzer eine Routine zur Verarbeitung von Bereichsinformationen geschaffen hat. Die Form dieser Routine muß der von DTFDA entsprechen.

g) TYPEFLE
Dieser Operand ist erforderlich.

h) HDRINFO
Dieser Operand ist anzugeben, wenn Kennsatzinformationen auf SYSLOG ausgegeben werden sollen, die zu Dateien mit Standardkennsätzen gehören.

Zu den Makros selbst gibt es noch folgendes zu sagen:

Das Makro FEOV ist nur bei logischen Einheiten des Programmierers erlaubt. Bei logischen Systemeinheiten wird diese Funktion durch das Makro

```
SEOV  dateiname
```

bewerkstelligt. Eine Registernotierung an Stelle des Dateinamens ist hierbei nicht erlaubt. Man beachte ferner, daß bei Datenträgerende der Übergang zum nächsten Datenträger nicht automatisch erfolgt, wenn das physische IOCS benutzt wird.

Byte	*Bitstelle*	*Bedeutung*	*Zuweisung einer 0 durch:*	*Zuweisung einer 1 durch: (sowie Anmerkungen)*
2	0	Verkehrsbit	EXCP	Kanalende, wenn auf Bitstelle 5 des Bytes 2 eine 0 steht Geräteende, wenn auf Bitstelle 5 des Bytes 2 eine 1 steht
	1	Dateiende (EOF) Gültigkeitsfehler beim Universaltypensatz	EXCP	Durch/*oder durch/& Nur bei der IBM 3211
	2	Fehler kann nicht behoben werden (bei der EA)	EXCP	Keine Ausführung der EA
	3	Behandlung nicht behebbarer EA-Fehler	Benutzer	Benutzer Befehlszurückweisungen, Programmfehler und Bandfehler beenden stets das Programm. Wenn auf dieser Bitstelle eine 1 steht, übergibt das IOCS bei Vorliegen anderer Fehlerbedingungen die Ablaufsteuerung zurück an das Benutzerprogramm. Eine 0 bewirkt, daß ein Eingreifen des Maschinenbedieners erfolgen soll.
	4	Rückkehr nach Datenfehlern bei SEDZ	Benutzer	Benutzer Steht auf dieser Bitstelle eine 1, so übergibt das IOCS die Programmablaufsteuerung zurück an das Benutzerprogramm, wenn ein Datenfehler auftritt. Bei einer 0 wird um ein Eingreifen des Maschinenbedieners nachgesucht.
	5	Festhaltung der Bedingung „Geräteende"	Benutzer	Benutzer Bei einer 0 wird das Geräteende vernachlässigt, bei einer 1 wird der Bitstelle 0 dieses Bytes erst bei Geräteende eine 1 zugewiesen.

Abb. 21.22. Die Kennzeichenbytes des Befehlssteuerblockes (CCB) (1. Teil)

Byte	*Bit-stelle*	*Bedeutung*	*Zuweisung einer 0 durch:*	*Zuweisung einer 1 durch: (sowie Anmerkungen)*
	6	Rückkehr bei Band- oder Plattenfehlern (beim Lesen und bei der Schreibprüfung), bei Kartenstanzfehlern und nach Fehlern bei der IBM 3211	Benutzer	Benutzer Bei 1 übergibt das IOCS die Programmablaufsteuerung an das Programm zurück, wenn ein Fehler auftritt; bei einer 0 wird ein Bedienereingriff angefordert.
	7	Fehlerroutine des Benutzers	Benutzer	Benutzer Bei einer 1 ist vom Benutzer die Behandlung von Gerätefehlern vorgesehen; bei einer 0 werden die Fehlerroutinen des IOCS verwendet.
3	0	Datenfehler im Kennzeichnungsfeld bei einer SEDZ Druckfehler bei der IBM 3211	EXCP	Datenfehler im Kennzeichnungsfeld. Gleichzeitig steht auf der Bitstelle 2 des Bytes 2 eine 1 und auf der Bitstelle 3 des Bytes 3 eine 0. Druckfehler bei der IBM 3211
	1	Spurenüberlauf bei einer SEDZ (Spurenüberschreitung) Druckfehler bei der IBM 3211	EXCP	Spurenüberlauf (Spurenüberschreitung) Druckfehler (Ausrüstung) bei der IBM 3211
	2	Zylinderende bei einer SEDZ Zeilenpositionsfehler bei der IBM 3211	EXCP	Zylinderende Gültigkeitsfehler, Datenfehler oder Ausrüstungsfehler
	3	Fehler in der Kartenstanzeinheit; Bandlesefehler; Fehler beim Lesen oder bei der Schreibprüfung bei einer SEDZ; Datenfehler (beim Druck) bei der IBM 3211	EXCP	Operation nicht erfolgreich; gleichzeitig enthält die Bitstelle 2 des Bytes 2 eine 1. Bei einer SEDZ wird zwar der Bitstelle 2 des Bytes 2 ebenfalls eine 1 zugewiesen, doch unterbleibt die Zuweisung einer 1 zu dieser Bitstelle.

Abb. 21.22. Die Kennzeichenbytes des Befehlssteuerblockes (CCB) (2. Teil)

Byte	*Bit-stelle*	*Bedeutung*	*Zuweisung einer 0 durch:*	*Zuweisung einer 1 durch: (sowie Anmerkungen)*
	4	Ausnahmebedingung	EXCP	Bei der IBM 2540 liegt eine anormale Befehlsfolge vor. Bei SEDZ ist kein Satz gefunden worden.
	5	Kein Satz gefunden	Benutzer	Benutzer Bei einer 1 wird der Befehl wieder aufgenommen, wenn es zu einer Bedingung „kein Satz gefunden" kommt; wenn auf dieser Bitstelle eine 0 steht, wird die Bitstelle 4 des Bytes 3 auf 1 gesetzt und eine Rückkehr zum Benutzerprogramm eingeleitet.
	6	Lochung im Kanal 9 des Druckerlochbandes Schreibprüfungsfehler bei einer SEDZ	EXCP EXCP	Lochung im Kanal 9; wird nur gesetzt, wenn eine 1 auf der Bitstelle 5 des Bytes 2 steht Schreibprüfungsfehler
	7	Wiederaufnahme der Befehlskettung	Benutzer	Benutzer Bei einer 1 wird die Prozedur vom letzten Kanalbefehlswort (CCW) aus vorgenommen, bei einer 0 vom ersten CCW.

Abb. 21.22. Die Kennzeichenbytes des Befehlssteuerblockes (CCB) (3. Teil)

Der Inhalt der Kennzeichenbytes im Befehlssteuerblock ist in der Abb. 21.22. zusammengefaßt.

21.16. Ein Beispiel

Als Beispiel für die Verwendung des IOCS wollen wir ein Problem betrachten, bei dem das Einlesen von Dualkarten über einen Kartenleser erforderlich ist. Die eingelesenen Sätze sind in eine logisch fortlaufend organisierte Datei zu stellen; der Inhalt von jeweils fünf Karten soll einen Block bilden (der Blockungsfaktor ist also gleich 5). Der Inhalt der Spalten 4 und 5 der Karten soll als Schlüssel herangezogen werden. Die Sätze der aufzubauenden logisch fortlaufend organisierten Datei sollen außerdem anders angeordnet sein als die Kartensätze; zunächst soll der Inhalt der Spalten 1 bis 3, danach der Inhalt der Spalten 6 bis 20 erscheinen, daran soll sich der Inhalt der Spalten 4 und 5 anschließen; zum Schluß soll der Inhalt der Spalten 21 bis 80 folgen.

Da durch das logische IOCS des Betriebssystems DOS die Karten nur in der Betriebsart 1 gelesen werden können, müssen wir für das Kartenlesen das physische IOCS einsetzen. Zum Erstellen der indexsequentiellen Datei werden wir selbstverständlich das logische IOCS benutzen.

Da wir für das Kartenlesen sowieso unser eigenes Kanalprogramm schreiben müssen, sollten wir bei diesem gleich die Datenkettung vorsehen. Auf diese Weise können wir nämlich das Schreiben einer Reihe von MVC-Instruktionen zur Neuanordnung der Daten vermeiden.

Wir wollen uns in dieser Situation nicht mit der Behandlung von Fehlern befassen. Wenn es zum Auftreten eines Fehlers kommt, soll einfach der Job beendet werden.

Unser Programm ist in Abb. 21.23. aufgelistet.

Man beachte die CLC-Instruktion. Da /* und /& nicht erkannt werden, wenn Dualkarten gelesen werden, müssen wir unsere eigene Dateiendeprüfung vorsehen.

Unser Programm kann auf drei verschiedenen Wegen beendet werden:

1) Beim Kartenlesen tritt ein Fehler auf. Das physische IOCS fordert in diesem Fall einen Eingriff des Maschinenbedieners an.

2) Fehler beim Plattenspeicher.
   In den meisten Fällen wird es sich hierbei um einen Reihenfolgefehler handeln, der dadurch entsteht, daß sich die Eingabekarten nicht in der geforderten Reihenfolge befinden, d.h. nicht nach aufsteigenden Schlüsseln sortiert vorliegen.

3) Dateiendebedingung (EOF).
   Da wir in der Betriebsart 2 arbeiten, wird die Dateiendekarte mit /* oder /& nicht als solche erkannt. Konsequent haben wir deshalb in unser Programm eine Prüfung aufgenommen, die das Vorhandensein des dualen Äquivalentes von /*b, nämlich 0C00 1022 0000 testet (b bedeutet hier das Leerzeichen).

Bei den meisten Anwendungen wird sich der Programmierer selbstverständlich zum Einsatz von Routinen entschließen, mit denen er die verschiedenen Fehlermöglichkeiten behandeln kann.

Da die indexsequentielle Datei in aufsteigender Folge geladen werden muß, müssen die Eingabekarten ebenfalls in aufsteigender Reihenfolge des Schlüssels zugeführt werden.

```
         PRINT NOGEN
         START
P1       BALR  10,0
         USING *,10
         OPEN  D1
         SETFL D1
P2       EXCP  C1
         WAIT  C1
         CLC   D3(6),=X'0C0010220000'
         BE    P3
         WRITE D1,NEWKEY
         WAITF D1
         TM    D1C,X'FB'
         BZ    P2
P3       ENDFL D1
         CLOSE D1
         EOJ
D2       DS    CL812
D3       DS    CL160
C1       CCB   SYSIPT,C2,X'0000'
C2       CCW   X'22',D3,X'80',6
         CCW   X'22',D3+36,X'80',4
         CCW   X'22',D3+6,X'80',30
         CCW   X'22',D3+40,X'00',120
D1       DTFIS IOROUT=LOAD,CYLOFL=1,DSKXTNT=2,NRECDS=5,RECSIZE=160,      1
               KEYLEN=4,KEYLOC=37,IOAREAL=D2,WORKL=D3,VERIFY=YES,        2
               RECFORM=FIXBLK,DEVICE=3330
         ISMOD IOROUT=LOAD
         END   P1
```

Abb. 21.23. Beispielsprogramm für das physische IOCS

21.17. Die Supervisormakroinstruktionen

Die Dienste des Supervisors kann sich der Programmierer durch eine Reihe von Makroinstruktionen sichern. Wir wollen sie der Reihe nach besprechen.

1) COMREG

Das Makro

```
COMREG
```

bewirkt, daß die Adresse des linksbündigen Bytes des Bereiches für den Informationsaustausch ins Register 1 gestellt wird.

2) MVCOM

Das Makro

```
MVCOM ziel,länge,ausgang
```

kann zur Änderung des Inhalts des Bereiches für den Informationsaustausch benutzt werden. Nur die Bytes 12 bis 23 können dabei geändert werden. Der Operand ziel ist gleich der Position des linksbündigen Bytes der Bytes 12 bis 23, das als erstes der Änderung unterliegen soll. Der Operand länge sagt aus, wie viel Bytes geändert werden sollen. Sein Wert muß also zwischen 1 und 12 liegen. Der Operand ausgang bedeutet den Namen desjenigen Speicherplatzes, von welchem die Daten abberufen werden sollen, die die im Bereich für den Informationsaustausch gespeicherten er-

setzen sollen. Der dritte Operand kann auch in Registernotierung in der Form (0) oder (r) mit r = 2, 3, . . . , 12 vorliegen, wenn zuvor in das betreffende Register die Adresse der anzusprechenden Daten gestellt ist.

3) LOAD

Das LOAD-Makro besitzt folgende Form:

```
LOAD  phasenname [,adresse]
```

An Stelle des Phasennamens und der Adresse können auch Registernotierungen codiert werden. Für den Phasennamen ist die Angabe von (r) oder (1) möglich, für die Adresse die Angabe von (r) oder (0). Diejenige Phase, die durch den Phasennamen angesprochen wird, wird von der Bibliothek in den Hauptspeicher eingelesen. Wenn kein zweiter Operand angegeben ist, so wird die Phase so geladen, daß sie auf der Hauptspeicheradresse beginnt, die zum Zeitpunkt ihres Aufbereitens durch den Linkage Editor festgelegt wurde. Die Adresse der Eingangsstelle dieser Phase (ebenfalls zur Aufbereitungszeit festgehalten) steht nach der Ausführung des Makros im Register 1.

Wenn eine Adresse oder ein Register, das eine Adresse enthält, als zweiter Operand spezifiziert ist, so wird diese Adresse als Startadresse benutzt, d. h. die Phase wird in die Hauptspeicherplätze geladen, die mit dieser Adresse beginnen. In diesem Fall findet keine Verschiebung der Adreßkonstanten statt. Die Eingangsstelle wird jedoch verschoben und ihre Adresse nach wie vor ins Register 1 gestellt. Wenn (r) oder (1) niedergeschrieben wird, so gehört die Adresse zu einem acht Bytes langem Feld, indem der Phasenname steht. Die spezifizierte Phase braucht nicht unbedingt Instruktionen zu enthalten, sie kann ebenso gut nur aus Tabellen und anderen Daten bestehen.

4) FETCH

Das FETCH-Makro besitzt die Form

```
FETCH phasenname [,eingangsname]
```

An Stelle des Phasennamens können auch die Registernotierungen (r) oder (1) und an Stelle des Eingangsnamens die Registernotierungen (r) oder (0) verwendet werden. Die durch den ersten Operanden spezifizierte Phase wird in den Hauptspeicher eingelesen. (Die Phase wurde in der PHASE-Karte benannt, als das Programm katalogisiert wurde). Wenn der zweite Operand weggelassen ist, so wird die Programmablaufsteuerung zu der Eingangsstelle geführt, die zur Aufbereitungszeit (Linkage Editor-Lauf) spezifiziert wurde. Bei Angabe eines Eingangsnamens wird jedoch die Programmablaufsteuerung der aufgeführten Eingangsstelle der betreffenden Phase übergeben (das erfordert natürlich während der Codierung des Programmes die Niederschrift einer ENTRY-Vereinbarung). Bei Verwendung der Registernotierung (r) geht die Programmablaufsteuerung zu der Adresse über, die im zitierten Register steht.

FETCH unterscheidet sich vom LOAD-Makro in der Hinsicht, daß durch dieses Makro die angegebene Phase nach dem Laden sofort ausgeführt wird, während durch LOAD die Phase nur geladen wird.

5) CANCEL

Dieses Makro besitzt die Form

```
CANCEL
```

Irgendwelche Operanden werden nicht verwendet. Der gesamte Job (nicht nur der gerade ablaufende Jobstep) wird durch dieses Makro abgebrochen. Ein Speicherauszug ist hierbei nicht vorgesehen.

6) EOJ
Durch das Makro

```
EOJ
```

wird der Jobstep beendet, der gerade abläuft.

7) DUMP
Durch das Makro

```
DUMP
```

wird der Jobstep beendet. Außerdem bewirkt es einen Auszug der Register und des Hauptspeichers in sedezimaler Form; dieser wird über SYSLST gedruckt.

8) PDUMP
Durch das Makro

```
PDUMP adr1,adr2
```

wird ein Auszug der Register und des Hauptspeichers zwischen den Adressen adr_1 und adr_2 bewirkt. Die Operanden können in Form von Registernotierungen (r) oder als Adreßausdrücke angegeben werden. Nach dem Auszug fährt das Problemprogramm mit der nächsten Instruktion fort.

9) JDUMP
Das Makro

```
JDUMP
```

bewirkt einen Auszug des Supervisors sowie der Partition und beendet anschließend den Job.

Der Benutzer kann Unterbrechungsbehandlungsroutinen für vier Unterbrechungsklassen schreiben. Dafür sind zwei Makros vorgesehen, die die Codierung solcher Routinen wesentlich erleichtern:

1) STXIT
Dieses Makro besitzt die folgende Form:

```
STXIT typ,adresse,rettungsgebiet
```

Unter typ ist die Unterbrechungsklasse wie folgt anzugeben:

a) AB
Anormale Beendigung durch eine andere Bedingung als die durch die Makros CANCEL, DUMP oder JDUMP gegebene. Nach dem Eintritt in die Benutzerroutine enthält Register 0 einen Code, der die Ursache des Programmfehllaufes angibt.

b) IT
Unterbrechung durch den Intervallzeitgeber

c) PC
Unterbrechung durch Programmfehler

d) OC
Externe Unterbrechung (Unterbrechungstaste usw.)

Unter adresse wird die Adresse des Hauptspeicherplatzes festgelegt, zu der der Supervisor verzweigen soll, wenn es zu der zitierten Unterbrechung kommt. Unter rettungsgebiet wird die Adresse eines aus 9 Doppelwörtern (18 vollen Wörtern) bestehenden Hauptspeichergebietes verstanden, in den der Supervisor das alte PSW

(Programmstatuswort) und die Inhalte der Register 0 bis 15 (in dieser Reihenfolge) hineingestellt. Das Rettungsgebiet wird bereits vor der Übergabe der Programmablaufsteuerung an die Benutzerroutine gefüllt. Der Adressenoperand kann auch in der Form (r) oder (0) angegeben werden; beim Rettungsgebietoperanden sind auch die Registernotierungen (r) oder (1) möglich.

Diese Unterbrechungsspezifikationen bleiben solange wirksam, bis entweder das Jobende erreicht ist oder eine andere STXIT-Makroinstruktion ausgeführt wird. Die Ausführung eines STXIT-Makros ohne Adreß- und Rettungsgebietangabe, das also in der Form

```
STXIT typ
```

codiert ist, hebt die vorhergehende Spezifikation für den angegebenen Typ auf, setzt jedoch keine neue an ihre Stelle.

Das Auftreten eines Programmfehlers während der Ausführung einer PC-Routine führt zur Beendigung des Jobs. Das Auftreten einer Unterbrechung durch den Intervallzeitgeber während der Ausführung einer IT-Routine führt ebenfalls zur Jobbeendigung, das Gleiche gilt für das Auftreten einer externen Unterbrechung während des Ablaufes einer OC-Routine.

2) EXIT

Um nach Durchlaufen einer Unterbrechungsroutine (PC, IT oder OC) zu der Programmstelle zurückzuspringen, an der es zur Unterbrechung kam, muß das Makro

```
EXIT  typ
```

ausgeführt werden. Die Register werden dabei auf ihre alten Werte zurückgesetzt und ferner wird auch das PSW wieder geladen. Die Benutzerroutine kann das PSW und die Registerstände im Rettungsgebiet ändern; dabei sollte man bei Änderung des PSW äußerste Sorgfalt walten lassen.

Routinen des AB-Typs dürfen durch EXIT nicht abgeschlossen werden; für sie sind die Makros CANCEL, DUMP oder JDUMP anzusetzen.

Der Intervallzeitgeber kann durch das Makro

```
SETIME sekunden
```

gesetzt werden. Der Operand kann auch in Registerform (1) oder (r) mit r = 2, 3, ···, 12 vorliegen. Gleichgültig, welche Form er aufweist, er gibt die Sekunden an, die in den Intervallzeitgeber hineingesetzt werden.

Das Makro

```
TTIMER [CANCEL]
```

gibt im Register 0 die in einem Intervall verbliebene Zeit zurück, und zwar in Hundertstel Sekunden als Dualzahl. Wird der Operand CANCEL benutzt, so wird der Zeitrest durch 0 ersetzt und die bei STXIT (Typ IT) angegebene Routine wird angesprungen.

Durch das Makro

```
GETIME einheit[,zone]
```

wird der Zugriff zur Tageszeit erreicht. Der zweite Operand kann wie folgt spezifiziert werden:

a) LOCAL
Damit wird die Lokalzeit zurückgegeben

b) GMT
Damit wird die mittlere Greenwicher Zeit zurückgegeben

Fehlt der zweite Operand, so wird LOCAL unterstellt. Für den ersten Operanden kann geschrieben werden:

a) STANDARD
Zeitrückgabe im Register 1 in der Form OHHMMSSt (H Stunden, M Minuten, S Sekunden) und zwar als gepackte Dezimalzahl.
b) BINARY
Register 1 wird auf die Zahl der Sekunden gesetzt, die bis zur Auslösung des Makros am Tage verstrichen sind, ausgedrückt als Dualzahl.
c) TU
Der im Register 1 zurückgegebene ist ein Vielfaches von Zeitgebereinheiten von 1/300 Sekunden.
d) MICRO
Die Tageszeit wird in den Register 0 und 1 als 64 Bitstellen belegende Dualzahl zurückgegeben. Diese Dualzahl entspricht den Mikrosekunden, die an dem betreffenden Tage verstrichen sind.

Das Makro

```
RELEASE (SYSnnn,···,SYSnnn)
```

kann dazu benutzt werden, um eine oder mehrere logische Einheiten des Programmierers vom Jobstep freizugeben. Bei einem Mehrprogrammbetrieb kann damit erreicht werden, daß die betreffenden Einheiten für Zuweisungen zu anderen Partitions bereits dann bereitstehen, wenn der Jobstep noch nicht beendet ist.

21.18. Speichermakros

Der SIZE-Parameter im EXEC-Statement der Jobsteuersprache teilt den virtuellen Speicher, der einer Partition zugeteilt ist, in zwei Teile auf. Wird SIZE=nnnn angegeben, so wird der gesamte Speicher jenseits von nnnn Bytes nach Partitionbeginn reserviert zum Gebrauch durch das GETVIS-Makro. Bei SIZE=AUTO wird der Speicher nach der in den Bytes 40 bis 43 des Bereiches für den Informationsaustausch für die GETVIS-Makroinstruktion bereitgestellt. Fehlt dieser Parameter jedoch, so gilt der gesamte Speicherraum einer Partition als vom Programm belegt und steht für die GETVIS-Makroinstruktion nicht mehr zur Verfügung.

Durch das Makro

```
GETVIS  ADDRESS=name1,
        LENGTH=name2
       [,PAGE={NO }]
              {YES}
```

wird ein virtueller Speicherblock zur eigenen Verwendung des Programmierers bereitgestellt. Der erste Operand kann auch in Registernotierung (1) oder (r) vorliegen, mit r = 2, 3, ···, 12. Die Adresse des virtuellen Speichergebietes, das zur Verfügung gestellt wird, wird ins Vollwort mit der Adresse name2 (Vollwort) zu finden ist. Bei PAGE = YES beginnt das bereitgestellte Speichergebiet an einer Seitenbegrenzung.

Der unter name2 spezifizierte Umfang wird vom System auf ein Vielfaches von 128 aufgerundet.

Das Operand ADDRESS kann fehlen; es wird dann ADDRESS=(1) unterstellt. Fehlt der Operand LENGTH, so gilt standardmäßig LENGTH=(Ø). Bei fehlendem PAGE-Operanden wird PAGE=NO angenommen.

Ein einmal zugeteiltes Speichergebiet kann wieder im ganzen oder nur stückweise freigegeben werden, indem das Makro

```
FREEVIS  ADDRESS=name1,
         LENGTH=name2
```

eingesetzt wird. Bei fehlendem ADDRESS-Operanden gilt standardmäßig ADDRESS=(1), bei fehlendem LENGTH-Operanden standardmäßig LENGTH=(0).

Bei GETVIS wird im Register 15 ein Code zurückgegeben. Es gibt folgende Codes:

0	Erfolgreiche Beendigung
4	Das zugeteilte Gebiet ist in Ordnung
8	Das Programm läuft im Realmodus (es gibt also keinen virtuellen Speicher)
12	Entweder ist die Angabe unter LENGTH negativ oder der für GETVIS zur Verfügung stehende Speicherraum ist bereits restlos ausgeschöpft

Der durch FREEVIS im Register 15 zurückgegebene Code besitzt folgende Bedeutung:

0	Erfolgreiche Beendigung
8	Programm läuft im Realmodus
12	name1 befindet sich nicht im GETVIS-Gebiet oder name2 ist kein Vielfaches von 128
16	name2 ist negativ oder name1 plus name2 überschreitet die Grenzen des GETVIS-Gebietes

21.19. Programmverbindungen

Im Abschnitt 20.21. haben wir die Programmverbindungen beschrieben. Als Hilfe zu ihrer Realisierung gibt es drei Makros:

CALL	$\left\{\begin{matrix}\text{eingangsname}\\(15)\end{matrix}\right\}[,(\text{arg}_1,\ldots,\text{arg}_n)]$
SAVE	$(r_1\ [,r_2]\ )$
RETURN	$(r_1\ [,r_2]\ )$

Aus dem CALL-Makro wird z. B. der folgende Code erzeugt:

```
         CNOP   2,4
name     L      15,=V(eingangsname)
         LA     14,*+6+4*n
         BALR   1,15
         DC     A(arg1,...,argn)
         ORG    *-4
         DC     X'8Ø'
         ORG
```

Es fehlt natürlich die L-Instruktion, wenn (15) als erster Operand benutzt wird.

Aus dem Makrostatement SAVE wird eine STM-Instruktion generiert, die die Registerinhalte von r_1 bis r_2 in das eingerichtete Standardrettungsgebiet wegspeichert. Fehlt r_2, so wird $r_2 = r_1$ angenommen. In gleicher Weise wird aus dem RETURN-Makro eine LM-Instruktion generiert, der die Instruktion

```
BR      14
```

folgt. Normalerweise werden bei SAVE und RETURN r_1=14 und r_2=12 gesetzt.

Aufgabengruppe 59

1) Schreibe ein Programm zum Einlesen eines Kartensatzes. Enthält die UPSI-Bitstelle 0 eine 1, so soll der Inhalt der Karten über SYSLST gedruckt werden. Ist der UPSI-Bitstelle 1 eine 1 zugewiesen, so soll der Inhalt der Kartenspalten 1 bis 76 in eine neue Karte gestanzt werden; in die Kartenspalten 77 bis 80 der neuen Karte ist außerdem die laufende Nummer der Karten 0001, 0002 usw., zu stanzen.

2) Man betrachte die Schleife

```
L       3,=F'1ØØØØØØ'
.       .
.       .
.       .
.       .
A       4,=F'1'
BCT     3,*-4
```

Schreibe ein Programm, da sich unmittelbar vor dem Eintritt in die Schleife und unmittelbar nach dem Normalausgang aus der Schleife die Zeit vom Zeitgeber holt, der deshalb zu Programmbeginn auf einen Anfangswert gesetzt werden sollte.

Berechne anschließend die Zeit in Sekunden und drucke sie aus! Wir erhalten damit die Zeitdauer, die für die Ausführung von je 1 Million (10^6) Instruktionen A und BCT erforderlich ist.

3) Ändere das Programm zu 2) so ab, daß in der Schleife zwei gleiche Additionsinstruktionen ablaufen. Wir erhalten jetzt die Zeit, die für 2 Mill. Additionsinstruktionen und 1 Mill. BCT-Instruktionen erforderlich ist.

Füge nun beide Schleifen zu einem einzigen Programm zusammen und drucke nunmehr die Zeiten aus, die zur Ausführung einer einzigen Additions- bzw. einer einzigen BCT-Instruktion erforderlich sind.

4) Verwende das Makro PDUMP, um die Inhalte der EA-Gebiete und der Inhalte von E30 bis E41 einschließlich auszugeben (Programmbeispiel 4 in Abb. 21.12.). Laß das PDUMP-Statement hinter jedem GET-Statement ablaufen.

5) Schreibe ein Programm, das den Inhalt des Gebietes ausgibt, das vom Beginn des Problemprogrammes bis zum Ende der laufenden Phase reicht. Die Ausgabe soll unmittelbar vor dem Abschluß der Programmausführung erfolgen. Die Verwendung des PDUMP-Makros soll beim Programm 4 in Abb. 21.12. gezeigt werden.

6) Wir wollen Karten einlesen können, die auf den Spalten 1 bis 6 den Wert von i in der Form .xxxxxx enthalten (der Dezimalpunkt ist jedoch nicht mit abgelocht) und auf den Spalten 7 bis 9 den Wert von n in der Form xxx.

Für jede eingelesene Karte ist eine neue Karte zu stanzen, die neben i und n auch den Wert von a in den Spalten 10 bis 20 aufweist.

Zur Berechnung von a dient die Formel

$$a = (1+i)^n$$

Der Wert von a ist mit 6 Dezimalstellen zu stanzen. Für das Lesen und Stanzen ist ein Programm in der AL zu schreiben; für die Berechnung ist ein Unterprogramm in der FORTRAN-Sprache zu schreiben.

7) Die Aufgabe 6) ist so zu modifizieren, daß statt des Unterprogrammes in der FORTRAN-Sprache ein solches in COBOL gefordert wird.

8) Ändere die Aufgabe 6) wie folgt ab:

   a) Ein PL/1-Hauptprogramm ist zu schreiben, das keine Verarbeitungsschritte enthält, jedoch das Programm in der AL aufruft, das Karten liest usw. Danach ist ein PL/1-Unterprogramm aufzurufen, das für die Berechnung zuständig ist und schließlich ist noch ein weiteres Unterprogramm in der AL aufzurufen, das das Stanzen bewirkt.
   b) Die erforderlichen Unterprogramme sind zu schreiben.

9) Unter Bezugnahme auf die Aufgabe 5 der Aufgabengruppe 58 ist ein Programm zu schreiben, das Karten einliest, die wie folgt abgelocht sind:

   a) Spalte 1 : 4
      Spalten 2 bis 9 : Kundennummer
      Spalten 10 bis 63 : Namen und Adresse des Kunden
      Spalten 64 bis 80 : leer

   b) Spalte 1 : 5
      Spalten 2 bis 9 : Kundennummer
      Spalten 10 bis 36 : Filiale, Stadt, Land
      Spalten 37 bis 46 : Kontenstand
      Spalten 47 bis 80 : leer

   Aus je zwei gelesenen Karten ist ein 100stelliger Bandsatz zu erstellen. Es soll dabei vorausgesetzt werden, daß die Eingabekarten nach dem Inhalt der Spalten 2 bis 9 sortiert sind.

10) Ändere die Aufgabe 9 wie folgt ab:

   Es ist eine indexsequentiell organisierte Datei zu erstellen, die 3fach geblockt ist. Die Kundennummer soll als Schlüssel benutzt werden.

11) Programmiere das Problem der Aufgabe 5 von Aufgabengruppe 58 neu und benutze dabei die durch das Programm der Aufgabe 9) erstellte Datei.

12) Schreibe ein Programm zum Lesen von Kundenkarten wie die in Aufgabe 9) beschriebenen und ersetze mit ihnen existierende Sätze in der in dexsequentiell organisierten Kundendatei (siehe Aufgabe 10)) bzw. nimm sie neu in die Kundendatei auf.

13) Regional organisierte Dateien sind dann besonders nützlich, wenn mit umfangreichen Matrizen gearbeitet werden muß. Schreibe ein Programm, durch das die Elemente von Matrizen in den Hauptspeicher eingelesen werden können, die auf Lochkarten wie folgt abgelocht sind:

   Spalten 1 und 2 : i
   Spalten 3 und 4 : j
   Spalten 5 bis 12: Matrixelement a_{ij}

Nach Voraussetzung soll $m < 12$ und $n < 4$ sein, so daß eine Matrix aus nicht mehr als 48 Elementen besteht.

Auf der ersten einzulesenden Karte sollen die Zeilenzahl m und die Spaltenzahl n der betreffenden Matrix abgelocht sein. Darauf folgen die $m \cdot n$ Karten mit den Matrixelementen; sie sind nicht sortiert. Nach dem Einlesen aller Karten sollen die Matrixelemente in eine regional organisierte Datei auf einem Plattenspeicher in der Reihenfolge $a_{11}, a_{12}, \ldots, a_{m,n-1}, a_{m,n}$ gespeichert werden. Auf die Verwendung aufgezeichneter Schlüssel soll verzichtet werden.

14) Schreibe ein Programm, mit dessen Hilfe die durch das Programm von Aufgabe 13 weggespeicherte Matrix vom Plattenspeicher in den Hauptspeicher zurückholt und zwar in der Reihenfolge $a_{11}, a_{12}, \ldots, a_{m-1,n}, a_{m,n}$.

22. Makrostatements im Betriebssytem OS

22.1. Die Programmierung der Eingabe/Ausgabe

Die Programmierung der Eingabe/Ausgabe ist bei den weitaus meisten Programmen fast identisch. Daraus kann gefolgert werden, daß die Arbeit des Programmierers beträchtlich vereinfacht werden kann, wenn standardisierte EA-Routinen für alle EA-Operationen eingesetzt werden. Die Einfügung solcher Routinen in Anwendungsprogramme ist an drei Voraussetzungen geknüpft:

1) Eine geeignete Routine muß in der Bibliothek SYS1.SVCLIB verfügbar sein.

2) Der Programmierer muß die Charakteristiken der Datenmenge so festlegen, daß das Betriebssystem bestimmen kann, welche Routine benutzt und von welchen Parametern sie abhängig gemacht werden muß.

3) Immer wenn der Programmierer wünscht, daß EA-Operationen ablaufen, muß er an Stelle der Maschineninstruktionen einen Pseudocode niederschreiben. Dieser Pseudocode oder Makrocode wird vom Assemblierer erkannt und in die geeigneten Statements umgesetzt, die die passende EA-Routine aufrufen.

Die IBM unterstützt sowohl die erforderlichen Makros als auch die dazugehörigen SVC-Routinen. Dem Programmierer bleibt es überlassen, die Charakteristiken der Datenmengen zu definieren. Diese Definitionen werden mittels eines anderen Makros vorgenommen, und zwar des DCB[1].

Nicht alle Parameter müssen im DCB-Makro definiert werden. Beim Eröffnen der Datenmenge können freigelassene Felder des Datensteuerblocks noch von Angaben im DD-Steuerstatement für die betreffende Datenmenge und sogar noch mit Informationen aus den Standardkennsätzen bei Magnetbändern und -platten gefüllt werden. Diese Technik ermöglicht es, daß die Eingabe/Ausgabe in beträchtlichem Maße geräteunabhängig ist, da der Gerätetyp, der für die Datenmenge vorgesehen ist, erst bei der Programmausführung genannt zu werden braucht und nicht schon beim Schreiben des Programmes. Natürlich gibt es auch Einschränkungen bezüglich der Geräteunabhängigkeit. Indexsequentiell und gestreut organisierte Datenmengen müssen auf SEDZ[2] residieren; bei einem Programm, das Vorschub und Zeilentransportbefehle für eine Datenmenge hinausschickt, muß diese Datenmenge einem Drucker zugeordnet sein. In gleicher Weise scheidet bei einer Blocklänge von 8000 Bytes aus, daß die betreffende Datenmenge auf einem Magnetplattenspeichergerät des Typs IBM 2311 residiert, es sei denn, daß die Spurenüberlaufeinrichtung (Spurenverbindung) installiert ist.

Wenn einmal der Datensteuerblock aufgesetzt ist, können im Problemprogramm Makrobefehle verwendet werden, die den Makrobefehlen des DOS sehr ähnlich sind. Meist werden die folgenden Makroinstruktionen benutzt:

a) OPEN

Diese Makroinstruktion liest und prüft oder schreibt Kennsätze auf Magnetbänder bzw. Plattenspeicher; außerdem dient sie zur Vervollständigung des Datensteuerblocks.

b) CLOSE

Diese Makroinstruktion verarbeitet die Nachsätze (Trailer labels) genauso, wie OPEN die Vorsätze (Header labels) bearbeitet. Außerdem bringt sie den Datensteuerblock auf den Zustand vor dem Eröffnen zurück und schließt die Datenmenge ab.

---

1) DCB ist die Abkürzung von „Data Control Block" (Datensteuerblock)

2) SEDZ ist die Abkürzung von Speichereinheiten mit direkter Zugriffsmöglichkeit

c) GET

Diese Makroinstruktion stellt dem Programm den nächsten logischen Satz zur Verfügung.

d) PUT

Diese Makroinstruktion bewirkt, daß der angegebene logische Satz zur Ausgabe bereitgestellt wird.

e) CNTRL

Diese Makroinstruktion ruft eine Steuerfunktion hervor, wie Zeilentransport bei Druckern, Fachauswahl bei Kartengeräten, Zurückspulen von Magnetbändern usw.

f) READ

Diese Makroinstruktion leitet die Datenübertragung von einem Gerät ein.

g) WRITE

Diese Makroinstruktion leitet die Datenübertragung zu einem Gerät ein.

h) CHECK bzw. WAIT

Diese Makroinstruktionen veranlassen die Zentraleinheit, auf die Beendigung einer EA-Operation zu warten.

Zusätzlich gibt es noch einige Makroinstruktionen, durch deren Hilfe Puffer für die Eingabe/Ausgabe bereitgestellt werden.

Jede EA-Operation gehört zu einer bestimmten Zugriffsmethode. Es gibt deren mehrere; wir wollen uns hier auf die folgenden fünf beschränken (VSAM behandeln wir in Kap. 23):

a) QSAM Sequentielle erweiterte Zugriffsmethode
(Queued Sequential Access Method)

b) BSAM Sequentielle Basis-Zugriffsmethode
(Basic Swquential Access Method)

c) QISAM Logisch fortlaufende erweiterte Zugriffsmethode
(Queued Indexed Sequential Access Method)

d) BISAM Logisch fortlaufende Bais-Zugriffsmethode
(Basic Indexed Sequential Access-Method)

e) BDAM Direkte Basis-Zugriffsmethode
(Basic Direct Access Method)

Da sich die englischen Kurzbezeichnungen inzwischen auch in Deutschland eingebürgert haben, wollen wir diese grundsätzlich beibehalten. Wir wollen fortan also von den Zugriffsmethoden QSAM, BSAM, QUISAM, BISAM und BDAM sprechen.

Tatsächlich zerfällt die Zugriffsmethode QISAM in zwei Grundmethoden: Die eine dient zum Laden, d.h. zur Erstellung einer Datei, und die andere zum Hinzufügen, Wiederauffinden und zum Aufdatieren (Fortschreiben, Ersetzen) der Sätze einer Datei. Im ersten Fall spricht man vom Lademodus (engl.: Load Mode) und im zweiten Fall vom Abtastmodus (engl.: Scan Mode).

Die erweiterten Zugriffsmethoden verwenden für die EA die Makroistruktionen GET und PUT, die Basis-Zugriffsmethoden die Makroinstruktionen READ, WRITE und CHECK.

Bei den bis jetzt diskutierten EA-Operationen braucht der Programmierer keine Kanalbefehlswörter (CCW) zu schreiben, da diese in den EA-Routinen enthalten sind. Diese Routinen, die kollektiv als "Eingabe/Ausgabe-Steuersystem" [1] bezeichnet werden, behandeln auch das Blocken und Entblocken von Sätzen, Fehlerbedingungen usw., sie befreien im allgemeinen den Programmierer von derartigen Überlegungen, so daß er sich ganz auf die logischen Funktionen seines Programmes konzentrieren kann. Diese Routinen werden unter der Sammelbezeichnung "logisches IOCS" zusammengefaßt. Eine andere Art von EA-Routine wird als "physikalisches IOCS" klassifiziert. Beim physikalischen IOCS muß der Programmierer die Kanalbefehlswörter (CCW) selbst codieren und ein Makro verwenden, das ihre Ausführung veranlaßt. Die Routinen des physikalischen IOCS setzen das Kanaladreßwort (CAW) auf, erarbeiten wahlweise Kennsätze und behandeln, ebenfalls wahlweise, Fehlerbedingungen. Das physikalische IOCS erfordert unzweifelhaft größere Anstrengungen auf Seiten des Programmierers. Allerdings erlaubt es dafür die großzügige Hantierung mit unüblichen EA-Vorgängen, für die im logischen IOCS keine Vorkehrungen getroffen sind.

Im ersten Teil dieses Kapitels behandeln wir das logische IOCS mit Ausnahme von VSAM, das wir im Kap. 23 abhandeln werden.

22.2. Der Datensteuerblock (DCB)

In der Abb. 22.1. sind die Operanden des DCB-Makros für die Zugriffsmethoden zusammengestellt, die wir bisher erwähnt haben. Der Landemodus der Zugriffsmethode QUISAM dient zur erstmaligen Erstellung einer indexsequentiell organisierten Datenmenge, der Abtastmodus wird für die Verarbeitung einer zuvor erstellten Datenmenge benötigt.

Durch den Operanden DSORG wird der Dateityp festgelegt, d.h. ob die Datei der konsekutiven, der indexsequentiellen oder der gestreuten Organisationsform unterliegt bzw. unterliegen soll. Entsprechend der vorgesehenen Zugriffsmethode sind auch die Parameter für diesen Operanden festzulegen: PS für die sequentielle, IS für die logisch fortlaufende und DA für die direkte[2]. Wird diesen Buchstabenkombinationen noch der Buchstabe U angehängt, so wird dadurch angezeigt, daß es sich um eine unbewegliche Datei handelt, um eine Datei also, die nicht verschieblich ist, die an feste Adressen gebunden ist. Somit werden durch PSU, ISU und DAU Datenmengen definiert, die nicht verschieblich sind. Im allgemeinen ordnet das OS einer Datenmenge den Speicherplatz auf externen Medien zu, den es für die betreffende Datenmenge als geeignet ansieht. Bei unbeweglichen Datenmengen beachtet es jedoch diesen Grundzug nicht. Bei unbeweglichen Datensätzen können deshalb z.B. die Schlüsselfelder oder die Datenfelder direkt das Kennzeichnungsfeld oder Teile desselben enthalten.

Durch den Operanden MACRF wird festgelegt, welche Arten von Makroinstruktionen für die betreffende Datei verwendet werden, d.h. GET, PUT, READ, WRITE usw. Die Kombination der Parameter von DSORG und MACRF bestimmt somit, welcher speziellen Zugriffsmethode die Datenmenge unterzogen wird. Diese beiden Operanden müssen deshalb unbedingt in jedem DCB-Makrostatement erscheinen. Die Niederschrift aller anderen Operanden ist freigestellt, da die betreffenden Informationen, falls sie benötigt werden, auch durch das DD-Statement zu der Zeit ergänzt werden können, zu der das Programm ablaufen soll; sie können sogar erst vom Programm selbst hinzugefügt werden, nur muß das natürlich vor der Eröffnung der Datei geschehen.

---

1) Im angloamerikanischen Sprachraum als „Input/Output Control System", abgekürzt IOCS, bezeichnet.

2) PS ist die Abkürzung von „physical sequential" (physikalisch fortlaufend), IS von „indexed sequential" (logisch fortlaufend) und DA von „direct access" (Direktzugriff)

Die Länge des Datensteuerblocks ist verschieden, sie hängt vom Gerätetyp ab, der im Objektprogramm für die betreffende Datei verwendet werden soll. Wenn DEVD angegeben ist, so enthält das Objektprogramm ein geräteabhängiges Gebiet innerhalb des Steuerblockes, das groß genug für den spezifizierten Gerätetyp ist. Wird der DEVD-Operand weggelassen, so wird stets der Parameter DA (direct access) angenommen und damit das größte überhaupt mögliche geräteabhängige Gebiet dem Steuerblock zur Verfügung gestellt.

Für den Operanden DEVD können die nachfolgenden Parameter aufgeführt und damit gewisse Geräteeinsatzmöglichkeiten geschaffen werden:

DEVD=DA	irgendein Gerät
DEVD=TA	irgendein Gerät außer SEDZ
DEVD=PR	irgendein Gerät außer SEDZ und Magnetbandgerät
DEVD=PC	Kartenstanzer
DEVD=RD	Kartenleser

Die Angabe des Operanden KEYLEN kann nur bei DEVD=DA erfolgen, DEN ist nur bei DA und TA möglich, Die Nennung von PRTSP erfordert das Vorhandensein eines Druckers (PR). Zu Kartenstanzern (PC), aber nur zu diesen können die Operanden MODE und STACK gehören, FUNC kann bei Vorliegen von PC als auch bei Vorliegen von PR auftreten.

Der Operand EROPT sagt dem Betriebssystem, daß beim Entstehen von EA-Fehlern diese zu akzeptieren (ACC) oder zu übergehen (SKP) sind bzw., daß das Programm abrupt (ABE) enden soll. Fehler bei der Magnetbandausgabe, die nicht beseitigt werden können, bewirken immer eine anomale Programmbedingung. Ist bei SYNAD eine Routine angegeben, so bekommt diese immer dann die Programmablaufsteuerung, wenn ein EA-Fehler passiert, die EROPT-Eintragung wird in einem solchen Falle ignoriert. Zu der bei EODAD angegebenen Adresse wird bei Feststellung des Dateiendes verzweigt. Die mittels EXLST festgehaltene Ausgangsliste ermöglicht den Anschluß von Routinen für die Behandlung von Benutzerkennsätzen, nichtstandardisierten Kennsätzen und von anderen irregulären Bedingungen. In diesem Buch wollen wir uns mit Ausgangslisten nicht beschäftigen.

Bei Magnetbändern wird die Aufzeichnungsdichte durch die Operanden DEN festgelegt. Es bedeuten:

DEN = 2	→	Aufzeichnungsdichte	800 bpi 1)
DEN = 3	→	Aufzeichnungsdichte	1600 bpi
DEN = 4	→	Aufzeichnungsdichte	6250 bpi

Bei Druckern wird durch die Eintragung PRTSP der normale Zeilentransport festgelegt. Bei Lochkarten wird durch MODE die Arbeitsweise festgehalten. Es bedeuten:

C	Lesen von Dualkarten
E	EBCDIC
R	Lesen mit Spaltenunterdrückung
O	Lesen optischer Markierungen

Der Operand STACK nennt das Ablagefach bei Kartengeräten. Durch FUNC schließlich wird die Verbreitungsart genannt. Es bedeuten:

1) bpi ist die Abkürzung von Bytes/Zoll

Operand	*Standard*	*Zugriffsmethoden*						*Quelle*		
		QSAM	BSAM	QUISAM Lademodus	QUISAM Abtastmodus	BISAM	BDAM	Programm	DD-Stat. der JSS	Kennsatz
DSORG=org	erforderlich	PS[U]	PS[U]	IS[U]	IS	IS	DA[U]	N	N	N
MACRF=typen	erforderlich	X	X	X	X	X	X	N	N	N
DDNAME=name	erforderlich	X	X	X	X	X	X	J	N	N
BLKSIZE=a | (a ≤ 32760)	erforderlich	X	X	X	—	—	X	J	J	J
LRECL=a | (a ≤ 32760)	erforderlich	X	X	X	—	—	—	J	J	J
RECFM=format	erforderlich	x	x	X	—	—	X	J	J	J
OPTCD=optionen	—	X	X	X	—	—	X	J	J	N
EROPT=ACC|SKP|ABE	ABE	X	—	—	—	—	—	J	J	N
EODAD=r	Abbruch	X	X	—	X	—	—	J	N	N
EXLST=r	—	X	X	X	X	X	X	J	N	N
SYNAD=r	EROPT	X	X	X	X	X	X	J	N	N
BFTEK=puffertyp	S	X	X	—	—	—	X	J	J	N
BFALN=F | D	D	X	X	X	X	X	X	J	J	N
BUFNO=a | (a ≤ 255)	2 oder 3	X	X	X	X	X	X	J	J	N
BUFL=a | (a ≤ 32760)	BLKSIZE	X	X	X	X	X	X	J	J	N
BUFCB=r	—	X	X	X	X	X	X	J	N	N
DEVD=gerätetyp	DA	X	X	—	—	—	—	N	N	N
KEYLEN=a | (a ≤ 255)	kein Schlüssel	—	X	X	—	—	X	J	J	J
DEN=2|3|4	Höchstwert	X	X	—	—	—	—	J	J	N
PRTSP=0|1|2|3	1	X	X	—	—	—	—	J	J	N
MODE=C|E|CR|ER|CO|EO	E	X	X	—	—	—	—	J	J	N
STACK=1|2	1	X	X	—	—	—	—	J	J	N
FUNC=fkt	—	X	X	—	—	—	—	J	J	N
NCP=a | (a ≤ 99)	1	—	X	—	—	—	—	J	J	N
LIMCT=a	—	—	—	—	—	—	X	J	J	N
NTM=a | (a ≤ 99)	kein Hauptindex	—	—	X	—	—	—	J	J	N
CYLOFL=a | (a ≤ 99)	0	—	—	X	—	—	—	J	J	N
RKP=a (0 ≤ a ≤ 32755)	0	—	—	X	—	—	—	J	J	N
MSHI=r SMHI=a | (a ≤ 65535)	nicht im Hauptspeicher	—	—	—	—	X	—	J	N	N
MSWA=r SMSW=a | (a ≤ 65535)	minimales Gebiet	—	—	—	—	X	—	J	N	N

10 28 36 43 50 57 64 71 78 85 92

Zeichenerklärungen:

X	Angabe möglich	a	absoluter Ausdruck
—	Operand wird für die betreffende Zugriffsmethode nicht benötigt	J	Aus dieser Quelle verfügbar
[U]	Wird U geschrieben, so gilt die Datenmenge als nicht verschieblich	N	Aus dieser Quelle nicht verfügbar.
r	relativer Ausdruck		

Abb. 22.1. Operanden des Makros DCB

D	Datenschutz; das Stanzen in bereits mit Lochkombinationen versehene Spalten ist nicht möglich
I	Daten werden gestanzt und gedruckt
P	Daten werden gestanzt
R	Daten werden gelesen
T	Zweizeilendruck; bei Weglassung Druck auf allen 25 Zeilen
W	Daten werden gedruckt
X	Eine zugeordnete Datei wird zum Stanzen und Drucken eröffnet

Folgende Funktionskombinationen können angegeben werden:

```
I  P   PW   PWXT
R  RP  RPD  RW  RWT  RWP  RWPXT  RWPD  RWPXTD
W  WT
```

Durch den Operanden BLKSIZE wird die maximale Blocklänge festgehalten; LRECL nennt die maximale Satzlänge. Dabei wird LRECL nur bei geblockten Sätzen benötigt. RECFM nennt die Satzart, diese Angabe muß auf die Zugriffsmethoden zugeschnitten sein. Der Operand OPTCD hält Optionen fest, die bei den einzelnen Zugriffsmethoden verfügbar sind.

Der Operand NCP gibt an, wieviel READ- bzw. WRITE-Statements maximal auftauchen können, bevor es zu einem CHECK-Makro für den betreffenden Datensteuerblock kommt. Damit ist auch die maximale Anzahl von Kanalprogrammen spezifiziert. Ein Puffer wird für jede Operation benötigt.

LIMCT nennt die Anzahl der Spuren oder Sätze (abhängig von der OPTCD-Eintragung), die nach einem Satz mit einem bestimmten Schlüssel durchsucht werden sollen. Fehlt dieser Operand, so wird nur die Orginalspur durchsucht.

Durch NTM wird die Maximalzahl der Spuren im Zylinderindex genannt, die erreicht sein muß, bevor es zum Aufbau eines Hauptindex kommt. Der Operand CYLOFL gibt die Zahl der Spuren für den Überlaufbereich in jedem Zylinder an. Unter RKP wird die relative Position (relativ zu 0) des ersten Bytes des im Satz eingebetteten Schlüssels verstanden; RKP=0 besagt, daß der Schlüssel auf der ersten Satzstelle beginnt. Die beiden Operanden MSWA und SMSW nennen Adresse und Länge des Arbeitsgebietes, das beim Hinzufügen eines Satzes zu einer indexsequentiell organisierten Datenmenge benötigt wird. Für Sätze im V-Format werden diese Eintragungen benötigt. Bei Sätzen im F-Format steigt die Verarbeitungsgeschwindigkeit an, wenn ein Gebiet definiert wird, das groß genug ist, um alle Teile aller Sätze einer Spur aufzunehmen; sind die Sätze geblockt, so sollte das Gebiet alle Blöcke einer Spur plus einen logischen Satz aufnehmen können. Die Operanden MSHI und SMSI dienen zur Festlegung eines Gebietes, das während der Verarbeitung den Hauptindex höchster Stufe im Hauptspeicher beinhaltet. Auch diese Maßnahme erhöht die Verarbeitungsgeschwindigkeit. Je größer das Gebiet im Verhältnis zum Gesamtumfang des Index ist, um so höher ist auch der Geschwindigkeitszuwachs.

22.3. Zugriff zum Datensteuerblock (DCB)

Der Datensteuerblock stellt nichts weiter als eine Tabelle von Konstanten dar. Da Bezugnahmen auf den DCB erst nach dem Eröffnen erfolgen, kann das Benutzerprogramm Felder im DCB solange modifizieren, solange die Datenmenge nicht eröffnet ist. Zur bequemen Bezugnahme auf die Felder des DCB ist ein Makro vorgesehen. Es lautet:

```
DCBD   DSORG=typ,DEVD=gerät
```

Als Parameter für DSORG ist einer der folgenden anzugeben:

QS	für QSAM
BS	für BSAM
PS	für QSAM und BSAM
IS	für QUISAM und BISAM
DA	für BDAM
LR	allein für LRECL

Für gerät ist eine der folgenden Angaben zu machen:

DA	für SEDZ
TA	für Magnetbandgeräte
PR	für Drucker
PC	für Kartenstanzer
RD	für Kartenleser

Der Parameter g e r ä t muß dem Parameter beim DEVD-Operanden im DCB-Makro gleichen. Durch das Makro DCBD wird ein Pseudoabschnitt (DSECT) mit dem Namen IHADCB erzeugt. In diesem haben alle Eintragungen Namen, die mit DCB beginnen; an diese drei Buchstaben ist der Name des betreffenden Feldes im DCB gekettet. Das heißt z.B., daß der im Pseudoabschnitt vorkommende Name DCBLRECL dem Feld LRECL im DCB entspricht. Besitzen die Felder im DCB Namen, die über fünf Buchstaben umfassen, so werden diese abgeschnitten. Somit bezieht sich der Name DCBBLKSI auf das Feld BLKSIZE. Nur ein DCBD-Makro kann in jeder Übersetzung benutzt werden, doch können ja Felder für mehr als einen Gerätetyp und mehr als eine Zugriffsmethode erzeugt werden. Dazu braucht man bloß die entsprechenden Parameter zu einer Liste zusammenzufassen und in Klammern zu stellen. Somit besagt also, daß das Makro

```
DCBD   DSORG=(QS,IS),DEVD=(TA,DA)
```

einen Pseudoabschnitt generiert, in dem Namen vorhanden sind, die sich auf alle Felder der Datensteuerblöcke von QSAM und von ISAM beziehen. Für die Zugriffsmethode QSAM wird z.B. der Name DCBDEN erzeugt, da zusätzlich FA spezifiziert wurde.

Der Operand DEVD kann weggelassen werden; in diesem Fall wird DA angenommen, allerdings nicht, wenn bei DSORG einer der Parameter QS, BS oder PS angegeben ist. Dann lautet der komplette Operand

```
DEVD=(DA,TA,PR,PC,RD)
```

Die Spezifizierung

```
DSORG=LR
```

bewirkt, daß nur der Name DCBLRECL generiert wird. Das kann durchaus sinnvoll sein, da die Länge eines gerade gelesenen Satzes in das Feld LRECL gestellt wird und eine Bezugnahme auf dieses Feld die einzige Möglichkeit ist, zur Satzlänge zuzugreifen. Das Schlüsselwort DEVD hat keinen Einfluß bei DSORG=LR.

22.4. Puffergebiete bei den erweiterten Zugriffsmethoden

Die erweiterten Zugriffsmethoden, von denen wir QSAM und QUISAM studieren werden, schließen drei Verfahren ein, durch die der Programmierer sich Puffergebiete für die

Dateien beschaffen kann. Was die Puffer selbst anbelangt, so sind für ihre Betrachtung und für das Verständnis ihrer Funktion nur vier Punkte heranzuziehen:

1) Die Anzahl der Puffer,
2) die Länge jedes einzelnen Puffers,
3) die Hauptspeicheradresse jedes einzelnen Puffers,
4) die Begrenzungszuordnung (Wort- oder Doppelwortgrenze).

Tatsächlich braucht der Programmierer sich nicht darum zu kümmern, wo die Puffer im Hauptspeicher liegen, da er niemals einen Puffer zu adressieren hat; in irgendeiner Weise muß er jedoch den IOCS-Routinen mitteilen, wo sie die benötigten Puffergebiete finden können. Kurz auf einen Nenner gebracht: Nur die ersten beiden Punkte brauchen den Programmierer zu interessieren.

Beim ersten Verfahren zur Beschaffung von Puffern verwendet man die Makroinstruktion GETPOOL. Dieses Makrostatement kann in einer der folgenden Formen auftreten:

```
GETPOOL   dcb,anzahl,länge
GETPOOL   dcb,(Ø)
```

Hierbei ist dcb entweder die Adresse des Datensteuerblocks DCB, für den der Pufferpool[1] eingerichtet werden soll, oder (1). Verwendet man als Parameter die Angabe (1), so muß die Adresse des DCB zuvor ins Register 1 geladen werden. Der erste Operand kann sich auf kein anderes Register als auf das Register 1 beziehen. Der zweite Operand gibt im allgemeinen die Anzahl der Puffer an, die bereitgestellt werden sollen, während durch den dritten Operanden die Länge eines jeden Puffers in Bytes festgelegt wird. Man kann aber auch in die linke Hälfte des Registers 0 die Anzahl der Puffer laden und in die rechte Hälfte die Länge eines jeden Puffers. In diesem Fall muß man als zweiten Operanden die Registernotierung für das Register 0, also (0) niederschreiben.

Wenn das Makro GETPOOL ausgeführt wird, so wird damit erreicht, daß ein unbenutzter Teil des Hauptspeichers für den Pufferpool abgezweigt wird. Die auf diese Weise erhaltenen Puffergebiete können nur für die Datei verwendet werden, die mit dem betreffenden Datensteuerblock verknüpft ist. Wenn GETPOOL verwendet wird, so bleiben die DCB-Operanden BUFNO und BUFL unberücksichtigt, da im GETPOOL-Makro diese Angaben bereits enthalten sind. Bei Verwendung des DCB-Operanden BFALN wird dieser vom GETPOOL-Makro dazu benutzt, um das erste Puffergebiet auf eine Doppelwortgrenze (bei Angabe des Parameters D) oder auf eine Wortgrenze (bei Angabe von F) auszurichten, d.h. beginnen zu lassen. Wenn die Pufferlänge durch 8 teilbar ist, so sind damit auch alle anderen Puffergebiete in der gleichen Weise ausgerichtet. Es ist allgemein üblich, die Pufferlänge als Vielfaches von 8 anzugeben, um damit eine zweckmäßige Begrenzungsordnung für jedes Puffergebiet zu erzwingen, sogar wenn die Satzlänge nicht ein Vielfaches von 8 ist.

Wenn der DCB-Operand BUFCB niedergeschrieben ist, so wird er vernachlässigt. Das GETPOOL-Makro stellt die Adresse des Pufferpools auf diesen Platz.

Das GETPOOL-Makro muß vor dem Eröffnen des Datensteuerblockes ausgeführt werden. Nachdem der DCB abgeschlossen ist, muß der Hauptspeicherteil, der für den Pufferpool bereitgestellt war, wieder an das OS zurückgegeben werden. Das geschieht durch das Makro

```
FREEPOOL dcb
```

1) Unter einem Pool versteht man eine Gruppe von Speichergebieten, die dem gleichen Zweck dienen sollen. An Stelle von Pufferpool oder kurz Pool ist auch der Ausdruck „Puffervorrat" gebräuchlich.

Hierbei bedeutet dcb entweder die Adresse des Datensteuerblockes oder ist durch die Registernotierung (1) auszudrücken. Im letzten Fall muß jedoch das Register 1 zuvor mit der Adresse des Datensteuerblockes geladen werden.

Nachdem einmal dieser Teil durch FREEPOOL wieder freigegeben ist, kann er vom OS wieder verwendet werden, um damit Speicherplatzanforderungen seitens von GETPOOL oder anderer Makros zu befriedigen.

Der zweite Weg, Puffer einer Datei zuzuordnen, besteht darin, daß man die erforderlichen Maßnahmen von OS selbst durchführen läßt. Wenn zu der Zeit, zu der das OPEN-Makro hinausgeschickt wird, keine Puffer vorhanden sind, so erzeugen die Routinen der Zugriffsmethoden QSAM bzw. QISAM automatisch das Äquivalent zu der GETPOOL-Makroinstruktion. In diesem Fall muß der DCB-Operand BUFCB weggelassen werden. Unter diesen Aspekten werden jeder Datei zwei Puffergebiete zugeordnet mit Ausnahme derjenigen Dateien, die zu einem Kartengerät IBM 2540 gehören; für diese werden nämlich drei Puffergebiete reserviert. Diese Standardzuordnung gilt natürlich dann nicht mehr, wenn durch den Operanden BUFNO eine andere Anzahl von Puffergebieten festgelegt wird. Die Begrenzungszuordnung ist auf eine Doppelwortgrenze ausgerichtet, es sei denn, daß durch BFALN eine andere festgelegt ist. Wenn BUFL nicht aufgeführt ist, so setzt QSAM die Pufferlänge gleich der durch BLKSIZE spezifizierten Länge. QISAM dagegen setzt die Pufferlänge gleich der Summe aus den durch BLKSIZE und KEYLEN spezifizierten Längen, gleichgültig, ob hier durch BUFL eine andere Angabe gemacht wurde oder nicht.

Die dritte Methode kann dann und nur dann verwendet werden, wenn der Programmierer zu der Zeit, zu der er das Quellenprogramm schreibt, die genaue Anzahl und den Typ der Puffer bereits kennt. Das Makro

```
BUILD pooladresse,anzahl,länge
BUILD pooladresse,(Ø)
```

bewirkt, daß ein vorhandener (durch DS definierter) Hauptspeicher so gegliedert wird, daß er als Pufferpool verwendet werden kann. Der erste Operand ist entweder die Registernotierung (1), wodurch angedeutet wird, daß die Gebietsadresse zuvor ins Register 1 geladen wurde, oder er stellt die symbolische Adresse des Gebietes dar, das als Pufferpool dienen soll. Der zweite und dritte Operand haben die gleiche Bedeutung wie der zweite und dritte Operand beim GETPOOL-Makro und können deshalb auch in der gleichen Form geschrieben werden. Das zu gliedernde Gebiet muß die Länge

$$8 + \text{anzahl} \cdot \text{länge}$$

besitzen, wobei die Größen anzahl und länge die Angaben des BUILD-Makros sind. Die ersten acht Bytes dienen als Pufferpool-Steuerblock. Bei Verwendung des Makros BUILD bleiben die DCB-Operanden BUFL und BFALN unberücksichtigt, da der Programmierer das Gebiet und damit auch seine Begrenzungsordnung selbst festgelegt hat, während die Pufferlänge durch das BUILD-Makro spezifiziert ist. Die Operanden BUFNO und BUFCB müssen dagegen im DCB-Makrostatement erscheinen. Durch BUFCB ist die Adresse des Pufferpools, festgelegt und durch BUFNO die Anzahl der Puffer, die der Datei aus dem Pufferpool während der Eröffnungszeit zugeordnet worden sind.

Die Verwendung von BUILD ermöglicht die gemeinsame Benutzung eines Pufferpools durch mehrere Dateien; vorausgesetzt hierbei ist natürlich, daß die Gesamtzahl der Puffer innerhalb des Pools groß genug ist, um allen Anforderungen nach Puffern gerecht werden zu können. Nach dem Abschließen der einzelnen Datensteuerblöcke werden die Puffer an den Pool zurückgegeben.

Die Verwendung des BUILD-Makros ist dann besonders wertvoll, wenn der Programmierer z.B. weiß, daß von fünf in seinem Programm angesprochenen Dateien nie mehr als drei zur gleichen Zeit eröffnet sind. Er könnte dann ein Gebiet definieren, das der größten Blocklänge, die für irgendeine Datei erforderlich ist, angepaßt ist und das gleichzeitig genügend Platz für die Puffer bietet, die von den Dateien benötigt werden, die die häufigsten Pufferanforderungen hinausschicken.

Bei der Verarbeitung von Spannsätzen können wir das Makro

```
BUILDREC pooladresse,anzahl,länge,satzadresse,satzlänge
```

zur Gestaltung des Pufferpools benutzen. Mit diesen Makro wird ein Puffer gebildet, der einen vollständigen Satz aufnehmen kann. Die Satzlänge muß um 32 größer angegeben sein, da 32 Bytes für Steuerzwecke erforderlich sind.

Beim Schließen einer Datei wird der durch GETPOOL (1. Methode) bzw. durch das OPEN-Statement (2. Methode) angeforderte Speicherplatz nicht von allein wieder freigegeben. Wie bereits bei der 1. Methode erwähnt, kann dieser Platz durch das Makro

```
FREEPOOL dcb
```

an das OS zurückgegeben werden. Der Operand kann auch in Registernotierung zu (1) oder (r) niedergeschrieben werden, wobei r eines der Register 2 bis 12 spezifiziert. Wird dagegen BUILD benutzt (3. Methode), so werden die Puffer an den Pool zurückgegeben. Hier ist es ja der Benutzer, nicht das System, durch den ein Gebiet für Pufferzwecke eingerichtet wird. Da stellt sich eine solche Frage, ob Puffergebiete an das System zurückzugeben sind, überhaupt nicht.

In fast allen Fällen holt man sich in der Praxis durch das OPEN-Statement die benötigten Puffer und gibt den von ihnen beanspruchten Platz durch FREEPOOL wieder frei. GETPOOL und BUILD werden nur in Sonderfällen benutzt. BUILDREC wird dann benötigt, wenn in einem bestimmten Augenblick ein vollständiger Satz und nicht ein Satzsegment verarbeitet werden soll.

Um den Vorteil zu erkennen, der in der Verwendung von mehr als zwei Puffern liegt, wollen wir dazu einen speziellen Fall betrachten. Nehmen wir einmal an, daß in einem Programm nur selten Ausgabesätze aufgebaut werden und daß der Aufbau dieser Sätze sehr schnell, d.h. mit nur wenigen Operationen, vonstatten gehen kann. Nehmen wir außerdem an, daß, wenn überhaupt Ausgabesätze zusammengestellt werden, diese immer wieder kurz hintereinander anfallen. Dann kann es dazu kommen, daß beispielsweise etwa vier Blöcke fast gleichzeitig herausgeschrieben werden müssen. Verwenden wir in einem solchen Fall vier Puffer, so können alle oder fast alle Ausgabeoperationen überlappt ablaufen.

22.5. Die Zugriffsmethode QSAM

Die Zugriffsmethode QSAM wird fast immer dann eingesetzt, wenn die Sätze konsekutiv organisierter Dateien sequentiell verarbeitet werden müssen. Zu ihren Einsatz kommt es, wenn im DCB die Operanden DSORG = PS bzw. DSORG = PSU vorliegen und im MACRF-Operanden folgende Parameter stehen:

(G x [c])
(P x [c])
(G x [c], Py [c])

Die eckigen Klammern zeigen an, daß C zusätzlich benutzt werden kann. Für x und y können die Buchstaben M, L, T oder D niedergeschrieben werden. Es bedeuten:

G	Benutzung des GET-Makros
P	Benutzung des PUT-Makros
C	Benutzung des CNTRL-Makros
M	Übertragungsmodus (kurz M-Modus) [1]
L	Lokalisierungsmodus (kurz L-Modus) [2]
D	Datenmodus [3]
T	Ersetzungsmodus [4]

Der Operand OPTCD kann folgendes Aussehen haben:

```
OPTCD=B
OPTCD=W[B]
OPTCD=Z[B]
OPTCD=U
```

Jeder Operand, der den Buchstaben B enthält, kann nicht im DCB-Statement definiert werden, sondern muß im DD-Statement erscheinen. Die Buchstaben bedeuten:

B	Bei Magnetbändern wird das Dateiende als Datenträgerende angesehen, und es werden dadurch soviele Datenträger verarbeitet, wie Seriennummern im DD-Statement aufgeführt sind.
Z	Bei Magnetbändern wird auf eine reduzierte Fehlerbeseitigung zurückgegriffen. Diese Option sollte nur dann benutzt werden, wenn von einen Magnetband bekannt ist, daß es Fehler aufweist und daß der Benutzer willens ist, die fehlerhaften Sätze zu übergehen.
W	Gültigkeitsprüfung bei SEDZ.
U	Nichtverhinderung von Datenfehlern bei einem Drucker mit dem Universaltypensatz. Ein Datenfehler bewirkt, daß der bei SYNAD genannte Ausgang betreten wird.

Keine der genannten Optionen kann man als eine vorzugsweise benutzte bezeichnen.

Die beim Operanden RECFORM möglichen Eintragungen lauten:

```
U[T][A|M]
V[B][S][T][A|M]
F[B][S|T][A|M]
```

Wird der Operand RECFM weggelassen, so wird für VSAM standardmäßig

```
RECFM=V
```

angenommen.

Eckige Klammern zeigen wiederum an, daß die Verwendung der in sie eingeschlossenen Angaben freigestellt ist. Der Vertikalstrich bedeutet hingegen, daß nur die vor ihm stehende oder die hinter ihm stehende Angabe (aber nicht beide Angaben gleichzeitg) verwendet werden darf. Die einzelnen Buchstaben bedeuten:

---

1) Im angloamerikanischen Sprachraum als „Move Mode" bezeichnet
2) Im angloamerikanischen Sprachraum als „Locate Mode" bezeichnet
3) Im angloamerikanischen Sprachraum als „Data Mode" bezeichnet
4) Im angloamerikanischen Sprachraum als „Substitute Mode" bezeichnet

U	Sätze unbestimmter Länge
V	Sätze variabler Länge
F	Sätze fester Länge
S	bei V: Spannsätze bei F: Die Datenmenge enthält mit Ausnahme des letzten Blockes keine verkürzten Blöcke
T	Benutzung der Spurenüberlaufeinrichtung bei SEDZ
A	Steuerzeichen nach ASA
M	Steuerzeichen im Maschinencode

Im Operanden BFTEK kann S, E oder A angegeben sein. Als Standard gibt S, was die einfache Pufferung anzeigt. E steht für Austauschpufferung und A zeigt an, daß eine einen logischen Satz beinhaltende Schnittstelle für Spannsätze vorhanden ist. Werden BFTEK und BFALN gleichzeitig benutzt, so müssen sie aus der gleichen Quelle (siehe Abb. 22.1.) stammen.

Abgesehen von den Makros, die zur Pufferverwaltung dienen (GETPOOL, FREEPOOL und BUILD), können bei der Zugriffsmethode QSAM die folgenden Makros verwendet werden:

a) `OPEN (dcb,[(methode[,disp])],···)`

Hierbei bedeutet dcb die Adresse des DCB und unter methode kann eine der folgenden Angaben stehen:

INPUT	Eingabe
OUTPUT	Ausgabe
UPDAT (nur bei SEDZ)	Aufdatierung
RDBACK	Eingabe (Rückwärtslesen von Magnetbändern)

Durch UPDAT wird festgelegt, daß das Makro PUTX verwendet wird, um existierende Sätze auf den neuesten Stand zu bringen (aufzudatieren) oder um sie zu ersetzen. Die letzte Angabe disp kann eine der folgenden sein:

DISP
REREAD
LEAVE } bei Datenträgerende

Durch DISP wird festgelegt, daß die im DD-Statement festgelegte Disposition verwendet werden soll, wenn die Umschaltung von einem Datenträger auf einen anderen erfolgt. REREAD bestimmt, daß der Datenträger so positioniert wird, daß die Datei erneut verarbeitet werden kann. Durch LEAVE ist schließlich festgelegt, daß der Datenträger beim Feststellen eines EOV-Kennsatzes (Ende des Datenträgers) keine Positionierung erfahren soll. Die dritte Eintragung disp darf nur dann vorgenommen werden, wenn auch die zweite Eintragung methode vorhanden ist. Fehlt die dritte Eintragung, so wird DISP angenommen. Wenn die zweite Angabe weggelassen wird, so wird INPUT vorausgesetzt.

Mit einem einzigen OPEN-Makro können beliebig viele Dateien eröffnet werden. Verfolgen wir dazu einige Beispiele:

```
OPEN   (A,(INPUT),B,(OUTPUT))
OPEN   (A,,C,(OUTPUT,DISP))
```

b) `CLOSE (dcb1,[disp1],[dcb2,disp2]]···)`

Hierbei bedeutet dcb_i die Adresse des Datensteuerblockes DCB, und durch $disp_i$ wird genau wie bei OPEN die Positionierung des Datenträgers festgelegt, nur daß diese bei Vorliegen der EOF-Bedingung (Dateiende-Bedingung) in Aktion tritt. Mit einem einzigen CLOSE-Makro können beliebig viele Dateien abgeschlossen werden. Man betrachte dazu die Beispiele:

```
CLOSE (A,,B)
CLOSE (A,,C,DISP)
```

Es gibt folgende Dispositionsoptionen:

REREAD	Einstellung auf den Anfang der Datei
LEAVE	Einstellung auf das Ende der Datei
REWIND	Einstellung auf den Ladepunkt
DISP	Disposition wie im DISP-Parameter des DD-Statements beschrieben
FREE	(nur bei der Betriebssystemvariante VS2): Datenmenge und Gerät, auf dem sie residiert, sollen nicht mehr dem Programm zugeordnet bleiben; damit können sie von anderen Jobs benutzt werden.

Wenn keine Option angegeben ist, wird DISP angenommen. Beim Rückwärtslesen von Magnetbändern sind die Funktionen von REREAD und LEAVE gerade vertauscht. Die Registernotierung (r) mit r = 2, 3, · · · 12 kann an Stelle von dcb bzw. dcb_1, dcb_2, · · · sowohl in den OPEN- als auch in den CLOSE-Statements verwendet werden.

c) Bei MACRF=GM wird der nächste Satz bereitgestellt, wenn das Makro

```
GET   dcb,gebiet
```

codiert ist. Der erste Operand kann auch in Registerform zu (1) oder (r) geschrieben werden, wobei r eines der Register 2 bis einschließlich 12 ist. Der zweite Operand kann ebenfalls in Registerform zu (Ø) oder (r) erscheinen. Der bereitgestellte Satz wird im Arbeitsgebiet gefunden. Wird ein Magnetband rückwärts gelesen, so stellt gebiet die Adresse des rechtsbündigen Bytes des im Programm vorgesehenen Arbeitsgebietes dar. Bei Spannsätzen wird der gesamte Satz im Gebiet zusammengebaut und außerdem das Satzbeschreibungswort gesetzt; aus ihm kann die Länge ersehen werden.

Bei MAKRF=GD, einer nur bei Spannsätzen erlaubten Spezifikation, wird fast dieselbe Aktion wie beim Übertragungsmodus (GM) ausgeführt, nur daß das Satzbeschreibungswort aus gebiet ausgeklammert ist.

Bei MAKRF=GL lautet die Form des GET-Makros wie folgt:

```
GET   dcb
```

Es wird also keine Arbeitsgebietadresse benutzt. Die Adresse des nächsten verfügbaren Satzes kann dem Register 1 entnommen werden. Der Satz selbst verbleibt im Eingabepuffer. Bei Spannsätzen wird auf einmal nur ein Segment geholt, es sei denn, daß ein BUILTREC-Makro hinausgeschickt oder BFTEK=A angegeben wurde. In diesen Fällen wird der gesamte Satz zur Verfügung gestellt. Sollte die Satzlänge großer als 32756 Bytes sein, so muß LRECL=X codiert sein.

Wenn MACRF=GT angegeben ist, so ist das GET-Makro in der Form zu codieren, die wir für den Übertragungsmodus kennengelernt haben. Das Gebiet, das im zweiten Operanden genannt ist, wird dem Betriebssystem zur Verwendung als Puffer übergeben. Die Satzadresse wird ins Register 1 gestellt.

Der Übertragungsmodus (GM) ist die meist benutzte Form, doch der Lokalisierungsmodus wird ebenfalls recht häufig verwendet. Der Datenmodus und der Ersetzungsmodus sind in der Praxis sehr wenig zu finden.

d) Ein Satz wird für die Ausgabe bereitgestellt, wenn das Makro

```
PUT   dcb,gebiet
```

benutzt wird. Für beide Operanden sind die folgenden Registernotierungen möglich: (1) oder (r) für den ersten und (Ø) oder (r) für den zweiten Operanden, wobei r eines der Register 2 bis einschließlich 12 darstellt.

Bei MACRF=PM wird der in gebiet stehende Satz in einen Puffer übertragen, von dem aus er dann im geeigneten Augenblick geschrieben wird. Im Datenmodus (MACRF=PD) wird ein Spannsatz von gebiet geschrieben; das Feld PRECL im DCB (DCB PRECL im Pseudoabschnitt) muß dabei vor Codierung des Makros auf die gesamte Satzlänge gesetzt sein. Im Ersetzungsmodus ist unter gebiet die Adresse eines zu schreibenden Satzes zu übergeben; danach übernimmt das Betriebssystem dieses Gebiet als Puffer. Im Register 1 wird durch PUT die Adresse eines anderen Puffers gesetzt, in dem dann ein anderer Satz zusammengestellt werden kann.

Im Lokalisierungsmodus fehlt der zweite Operand. Durch Ausführung des PUT-Makros wird ins Register 1 die Adresse eines Puffers gestellt, in dem der Benutzer seinen Satz zusammenbauen kann. Der Satz selbst wird erst beim nächsten PUT-Statement bzw. beim CLOSE-Statement ausgegeben. Der Lokalisierungsmodus des PUT-Statements unterscheidet sich also wesentlich von dem des GET-Statements. PUT meldet nämlich erst die Adresse eines Gebietes zurück, in dem der Benutzer seinen Satz zusammenstellen kann.

e) Für eine Datei, die mit UPDAT eröffnet wurde, steht noch das Makro PUTX in einer der beiden nachstehenden Formen zur Verfügung:

```
PUTX  dcb
PUTX  (1)
PUTX  (r)
```

Dieses Makro bewirkt, daß ein Satz aufdatiert wird, d.h. den bisherigen Satz ersetzt, der durch ein GET-Statement im L-Modus eingelesen wurde. Diese Codierung ist natürlich nur für Dateien möglich, die auf einer SEDZ residieren.

Das Makro

```
PUTX   dcbaus, dcbein
PUTX   dcbaus, (Ø)
PUTX   (1), dcbein
PUTX   (1), (Ø)
```

bewirkt, daß ein Satz, der durch ein GET-Statement im L-Modus gelesen wurde, durch ein PUT-Statement im M-Modus ausgegeben wird. Der erste Operand spezifiziert die Adresse des DCB der Ausgabedatei, der zweite Operand die Adresse des DCB der Eingabedatei. Der DCB der Eingabedatei muß mit dem Operanden MACRF=(GL) und der DCB der Ausgabedatei mit MACRF=(PM) eröffnet werden. In vielen Fällen der Praxis kann diese Kombination sich als recht nützlich erweisen.

Die nachfolgenden Makros üben die verschiedensten Funktionen aus. Betrachten wir sie der Reihe nach:

a)
```
RELSE dcb
RELSE (1)
```

Dieses Makro bewirkt, daß der restliche Inhalt eines Eingabepuffers übergangen wird. Das nächste GET-Makro greift zum ersten logischen Satz des nachfolgenden Eingabepuffers zur Verfügung. Auf ungeblockte Sätze übt dieses Makro keine Wirkung aus.

b)
```
TRUNC dcb
TRUNC (1)
```

Die Ausführung der durch dieses Makro erzeugten Instruktions- bzw. Befehlsfolge bewirkt, daß der laufende Ausgabepuffer als gefüllt angesehen und daher geschrieben werden soll. Wenn TRUNC einem anderen TRUNC unmittelbar folgt oder bei ungeblockten Sätzen angewendet wird oder dann zum Tragen kommt, wenn ein Puffer bereits gefüllt ist, so bleibt dieser Makrobefehl ohne Effekt. RECFM=FBS verbietet die Benutzung dieses Makros.

c) Das Makro

```
FEOV  dcb [,REWIND|LEAVE]
```

bewirkt, daß die Bedingung „Datenträgerende" angenommen wird. Wird dieses Makro für die letzte Magnetbandrolle einer Datenmenge hinausgeschickt, so kommt es zum Betreten der Ausgangsroutine EODAD, die sonst nur für Eingabedateien vorliegt. REWIND bewirkt, daß das Magnetband bis zum Ladepunkt zurückgespielt wird; LEAVE beläßt das Band am Ende der Datenmenge auf dem betreffenden Datenträger.

d) Das Makro

```
CNTRL dcb,funktion
```

dient zur Realisierung der verschiedensten geräteabhängigen Funktionen. An Stelle von dcb kann auch eine der Registernotationen (1) oder (r) erscheinen, wobei r auf eines der Register 2 bis einschließlich 12 verweist. Für funktion können nachstehende Angaben gemacht werden:

SS,1 SS,2	Auswahl des Ablagefachs $\left\{\begin{matrix}1\\2\end{matrix}\right\}$ bei einem Kartenleser
SP,1 SP,2 SP,3	Zeilentransport um $\left\{\begin{matrix}1\\2\\3\end{matrix}\right\}$ Zeilen bei einem Zeilendrucker bzw. bei der IBM 3525
SK,c	Vorschub bis zum Kanal c bei einem Zeilendrucker oder bei der IBM 3525, wobei c die Werte 1 bis 12 einschließlich annehmen kann
BSM	Zurücksetzen des Magnetbandes bis hinter eine Bandmarke, dann Überspringen der Bandmarke in Vorwärtsrichtung
FSM	Vorwärtssetzen des Magnetbandes bis hinter eine Bandmarke, dann Überspringen der Bandmarke nach rückwärts
BSR,n	Zurücksetzen des Magnetbandes über n Blöcke hinweg; fehlt n, so wird n gleich 1 angenommen
FSR,n	Vorwärtssetzen des Magnetbandes über n Blöcke hinweg; fehlt n, so wird n gleich 1 angenommen.

e)
```
PRTOV dcb,zahl [,adresse]
```

Dieser Makrobefehl bewirkt, daß eine Prüfung vorgenommen wird, um festzustellen, ob im Kanal 9 oder im Kanal 12 des Vorschublochbandes beim Drucker eine Lochung vorhanden ist. Als zweiter Operand muß 9 oder 22 angegeben werden, um den zu

testenden Kanal bestimmen zu können. Fehlt die Angabe des dritten Operanden, so tritt automatisch ein Vorschub bis zu einer Lochung im Kanal 1 ein, wenn die Prüfung erfolgreich war. Wenn dagegen der dritte Operand niedergeschrieben wird, so erfolgt bei erfolgreichem Test eine Verzweigung zu der spezifizierten Adresse. Wenn eine Benutzerroutine geschrieben wird, so sollte DCBBUFNO auf 1 gesetzt werden, um damit sicherzustellen, daß alle anstehenden Druckoperationen abgeschlossen sind, wenn dieser Test vorgenommen wird. Die Seitenende-Bedingung wird dann entdeckt, wenn der Versuch unternommen wird, eine Zeile zu drucken, die auf die Zeile folgt, die durch eine Lochung im Kanal 12 oder im Kanal 9 markiert ist.

Wird zur angegeben Adresse verzweigt, so besitzen die einzelnen Register folgende Inhalte

0 1	nicht vorhersagbar
2 . .	unverändert
. 13	
14	Rückkehradresse vom IOCS
15	Adresse der Ausgangsroutine

Auch bei diesem Makro kann der erste Operand durch eine Registernotierung (r) ersetzt werden, wobei unter r wie üblich eines der Register 2 bis einschließlich 12 gemeint ist.

Die Benutzung der beiden unter d) und e) genannten Makros sollte nur mit Vorsicht erfolgen, da sie ein Programm geräteabhängig machen. Da PRTOV außerdem das Warten auf die Geräteendebedingung erfordert und nicht wie sonst üblich das Warten auf die Kanalendebedingung, ergibt sich allein daraus eine Verlangsamung des Programmes.

22.6. Die Zugriffsmethode QUISAM

Die Zugriffsmethode QISAM kennt zwei verschiedene Arbeitsweisen, den Lademodus (Load Mode) und den Abtastmodus (Scan Mode). Der Lademodus wird dann verwendet, wenn eine neue Datei erstellt oder eine bestehende ausgedehnt werden soll. Dabei müssen die Schlüssel neuer Sätze größer sein als die höchsten bisher in der Datei vorhandenen. Der Abtastmodus erlaubt das Wiederauffinden und die Aufdatierung von bestehenden Sätzen.

Beim Lademodus muß der MACRF-Operand entweder MACRF = PL oder MACRF = PM lauten; damit wird angezeigt, daß das PUT-Makro entweder im L-Modus oder im M-Modus verwendet wird. Im Abtastmodus können die Eintragungen beim MACRF-Operanden wie folgt lauten:

```
GM
GMSK
GMSI
GL
GLSK
GLSI
(GLSK,PU)
(GLSI,PU)
```

Die Buchstabenkombination GN besagt, daß GET-Statements im M-Modus auftreten, GL, daß solche im L-Modus vorliegen. SI sagt aus, daß Makro SETL mit der Satzkennzeichnung MBBZZKKS, SK, daß SETL mit dem Satzschlüssel vermutet wird. PU bedeutet die Verwendung des Makros PUTX. Bei QUISAM darf PUTX nur zum Zurückschreiben geänderter Sätze dienen.

Beim Operanden RECFM ist die Angabe von V, VB, F oder FB möglich.

Der Operand OPTCD wird nur bei der Erstellung einer Datenmenge benutzt. Er kann wie folgt codiert werden:

OPTCD = [I] [L] [M] [R] [U] [W] [Y]

Die einzelnen Buchstaben können in einer beliebigen Reihenfolge erscheinen. Sie bedeuten:

Y	Die Zylinderüberlaufbereiche werden benutzt; fehlt Y, so wird das für CYLOFL reservierte Gebiet nicht benutzt
W	Schreibprüfung
U	(nur bei Sätzen im F- oder FB-Format): Die Eintragungen für den Spurenindex werden auf einmal für die ganze Spur im Hauptspeicher zusammengestellt. Resultat: Verminderung der EA-Zeichen auf Kosten des Speicherbedarfs
M	Erstellung eines Hauptindex, wie unter NTM vermerkt ist
I	Verwendung von unabhängigen Zylinderüberlaufbereichen
L	Sätze, die durch lauter Einerbits im am weitesten links stehenden Byte gekennzeichnet sind, dürfen bei Bedarf aus der Datei entfernt werden (beim Hinzufügen neuer Sätze)
R	Es sind Vorbereitungen zu treffen für die Rückmeldung von Reorganisationskriterien

Die Angabe von R bewirkt, daß drei Felder des DCB laufend auf den neuesten Stand gebracht werden:

DCBORG1	Anzahl der vollen Zylinderüberlaufbereiche
DCBORG2	Anzahl der verbleibenden Spuren im unabhängigen Überlaufbereich
DCBORG3	Anzahl der READ- und WRITE-Statements, die sich auf einen Überlaufsatz beziehen, der nicht der erste Überlaufsatz einer Kette ist

In allen drei Feldern werden Dualzahlen im Umfang eines Halbwortes gespeichert.

Bei den GET- und PUT-Operationen ist die Länge des Arbeitsgebietes gleich der Summe aus der unter LRECL angegebenen Länge und der Schlüssellänge (unter KEYLEN) 1, wenn die Schlüssel nicht in den Sätzen eingebettet sind und die Schlüssel gelesen werden. Der Schlüssel belegt den linksbündigen Teil des Arbeitsgebietes und wird von den Daten gefolgt.

Der Parameter von BUFL wird stets durch einen im OPEN-Makro errechneten Wert ersetzt. Durch OPEN wird die erforderliche minimale Blocklänge bestimmt und mit der spezifizierten Länge verglichen. Ist diese groß genug, so wird sie durch die errechnete Länge ersetzt. Die für logisch fortlaufend organisierte Dateien erforderliche minimale Pufferlänge ist in der Abb. 22.2 aufgeführt.

Für die Verarbeitung von Sätzen gibt es die folgenden Makros:

OPEN
CLOSE
GET
PUT
PUTX
RELSE

Schlüsselposition RKP	Satzform RECFM	Satzlänge s LRECL	Blocklänge b BLKSIZE	Pufferlänge BUFL	
				Lademodus (Load)	Abtastmodus (scan)
0	F	l	s	8+f+b	f+10+b *)
0	V	4+l	4+s	8+f+b	f+10+b *)
≠0	F[B]	l	s·n	8+b	6+10+b
≠0	V[B]	4+l	4+s·n	8+b	2+10+b

Es bedeuten: l Anzahl der von Daten belegten Bytes
s Länge eines logischen Satzes
b Länge eines Blockes (Länge eines physikalischen Satzes)
n Blockungsfaktor
f Schlüssellänge

Anmerkung: *) Die Pufferlänge wird auf ein Vielfaches von 8 aufgerundet

Abb. 22.2. Pufferlängen für logisch fortlaufend organisierte Dateien

Die Datei muß durch die Angabe von OUTPUT eröffnet werden. Alle Makrostatements besitzen die gleichen Formen, die wir bereits bei der Besprechung von QSAM kennengelernt haben. Wir sollten uns jedoch folgendes unbedingt merken:

a) Bei OPEN ist nur die Option OUTPUT erlaubt
b) GET und PUT können nur im L-Modus und im M-Modus benutzt werden
c) PUTX wird nur zur Aufdatierung benutzt, nicht zur Erstellung neuer Dateien

Zusätzlich gibt es zwei andere Makros. Sie lauten:

```
SETL   dcb,code[,adresse]
ESETL  dcb
```

In Abwesenheit des Makros SETL wird die Datei von Beginn an bis zum Ende sequentiell auf der Basis der Schlüssel der Sätze verarbeitet. SETL kann nun dazu benutzt werden, um die Verarbeitung an anderer Stelle als beim ersten Satz zu starten. Für code können die folgenden Werte angegeben werden:

B	Die nächste Eingabeoperation beginnt am Dateianfang
I	Unter adresse ist eine echte Plattenspeicheradresse in der bekannten Form MBBZZKKS aufgeführt
K	Unter adresse ist der Schlüssel eines Satzes aufgeführt; die nächsten Eingabeoperationen beginnen mit dem Satz, dessen Schlüssel unter adresse aufgeführt ist
KH	Unter adresse ist ein Satzschlüssel genannt; die nächsten Eingabeoperationen beginnen mit dem Satz, dessen Schlüssel gerade noch kleiner als oder gleich dem angegebenen Satzschlüssel (generischer Schlüssel) ist
KC	Unter adresse ist eine Schlüsselklasse genannt; die nächsten Eingabeoperationen beginnen mit dem Satz, dessen Schlüssel gerade noch kleiner als oder gleich dem unter adresse genannten Schlüssel ist. Die rechtsbündigen Bytes von adresse können aus dualen Nullen bestehen; solche Bytes werden nicht mit in den Vergleich einbezogen.
BD, ID, KD, KDH, KCD	(nur für Sätze im F-Format ohne eingebettete Schlüssel): Der Buchstabe D zeigt an, daß nur der Datenteil des Satzes zu lesen ist. Puffer- und Arbeitsgebietslängen brauchen keinen Platz für den Schlüssel zu berücksichtigen

Nach der Ausführung eines SETL-Makros fährt die Verarbeitung sequentiell von diesem Punkt an fort, bis entweder das Dateiende erreicht ist oder einem ESETL-Makro begegnet wird. Nach dem ESETL-Makro kann ein neues SETL-Makro codiert werden.

Für dcb kann (1) oder (r) geschrieben werden, für adresse (0) oder (r); unter r ist eines der Register 2 bis einschließlich 12 zu verstehen.

Bei OPTCD = L wird ein Satz für die Entfernung vorgemerkt, indem im linksbündigen Byte des Satzes alle Bitstellen auf 1 gesetzt werden. Bei Sätzen im V- oder VB-Format wird FF (sedezimal) ins 5. Byte gestellt, d.h. in das erste Byte, das auf das vier Bytes lange Satzlängenfeld folgt.

1. Byte DCBEXCD1

Bitstelle	Bedeutung	Gesetzt durch
0	Untere Schlüsselgrenze nicht gefunden	SETL
1	Ungültige Kennzeichnung für einen Satz	SETL
2	Kein Platz gefunden, um einen Satz hinzufügen zu können	PUT (Lademodus)
3	Ungültige Statements: a) BUFL zu klein b) Es wurden zwei Makroinstruktionen SETL gegeben, ohne daß zwischen ihnen eine Makroinstruktion ESETL auftritt c) Unkorrektes Typenfeld bei der Makroinstruktion SETL	SETL
4	Unkorrigierbarer Eingabefehler	GET
5	Unkorrigierbarer Ausgabefehler	PUT oder CLOSE (Lademodus)
6	Bei der Eingabe kann der entsprechende Block nicht erreicht werden	GET
7	Überlaufsatz	PUT oder CLOSE (Lademodus)

2. Byte DCBEXCD2

Bitstelle	Bedeutung	Gesetzt durch
0	Reihenfolgefehler: Die Sätze werden nicht in ihrer richtigen Reihenfolge geladen	PUT (Lademodus)
1	Doppelter Satz (Duplikatsatz)	PUT (Lademodus)
2	Ein unkorrigierbarer Fehler wurde während der Ausführung von CLOSE entdeckt	CLOSE
3	Überlaufsatz (gilt nicht als Fehler)	GET
4 bis 7	Für zukünftigen Gebrauch freigehalten	----

Abb. 22.3. Die beiden Bytes des Datensteuerblockes bei der Zugriffsmethode QISAM, die die außergewöhnlichen Bedingungen registrieren

Bei auftretenden Fehlerbedingungen werden bestimmte Bitstellen von zwei Bytes des DCB auf 1 gesetzt. In Abb. 22.3 sind die Bedingungen zusammengestellt. Die bei SYNAD genannte Routine wird betreten, wenn die Bitstellen 4 bis 7 des Bytes DCBEXCD1 nicht alle auf Null stehen.

22.7. Einige Beispiele

In diesen Abschnitt wollen wir uns vier Programme ansehen, die die gleiche Aufgabe erfüllen wie die im Abschnitt 21.6. vorgestellten. Es wird ein Kartensatz eingelesen, und der Inhalt der einzelnen Karten als Sätze auf ein Magnetband geschrieben; dabei geht der Inhalt der Kartenspalten 73 bis 80 auf die Stellen 1 bis 8 des Magnetbandsatzes über, der Inhalt der Kartenspalten 1 bis 72 auf die Stellen 9 bis 80.

Wir haben diese Beispiele so ausgewählt, daß es uns möglich ist, die verschiedenen Codierungsmöglichkeiten zu zeigen. Wir wollen besonders den Unterschied zwischen dem M-Modus und L-Modus herausstellen. Wir wollen auch hier nicht unerwähnt lassen, daß ein DCB nicht in die ersten 16 Bytes eines Programmabschnittes gestellt werden kann. Außerdem muß ein Basisregister für einige möglichen Instruktionen im DCB verfügbar sein. In Abb. 22.4. wird ein Programm gezeigt, bei dem der Übertragungsmodus sowohl für die Eingabe als auch für die Ausgabe benutzt wird. Die FREEPOOL-Makros sind nicht unbedingt notwendig, denn das nachfolgende Jobende gibt automatisch alle Puffer frei. Das Makro SAVE ergibt bei der Auflösung die Instruktion

```
        STM     14,12,12(13)
```

Aus dem RETURN-Makro wird die Instruktionsfolge

```
        LM      14,12,12(13)
        LA      15,Ø
```

oder

```
        LM      14,12,12(13)
        BR      14
```

generiert.

```
*               PROGRAMM MIT ARBEITSGEBIETEN
*               ----------------------------
*
CDTE    CSECT
        USING *,11
E10     SAVE  (14,12)
        LR    11,15
        LA    14,RETTGEB
        ST    14,8(13)
        ST    13,4(14)
        LR    13,14
        OPEN  (KARTEIN,(INPUT),BANDAUS,(OUTPUT))
E11     GET   KARTEIN,E31
        MVC   E51,E31+72
        MVC   E52,E31
        PUT   BANDAUS,E50
        B     E11
```

```
E20      CLOSE (KARTEIN,,BANDAUS)
         FREEPOOL KARTEIN
         FREEPOOL BANDAUS
         L      13,4(13)
         RETURN (14,12),RC=0
KARTEIN  DCB    DSORG=PS,MACRF=GM,
                DDNAME=SYSIN,EODAD=E20
BANDAUS  DCB    DSORG=PS,MACRF=PM,
                DDNAME=TPUT,RECFM=FB,
                BLKSIZE=320,LRECL=80
E81      DS     CL80          ARBEITSGEBIET FUER DIE EINGABEDATEI
E50      DS     OCL80         ARBEITSGEBIET FUER DIE AUSGABEDATEI
E51      DS     CL8
E52      DS     CL72
RETTGEB  DC     18F'0'
         END    E10
```

Abb. 22.4. Programm mit Arbeitsgebieten

Man beachte, daß einige Informationen über die Kartendatei nicht im DCB-Statement genannt sind. Diese kommen von den Plattenkennsätzen, vorausgesetzt, daß die Karten im Eingabestrom für die Jobs liegen.

Abb. 22.5. verwendet den Lokaliserungsmodus für die Eingabe. Man beachte hierbei die Verwendung des Registers 1.

```
*                PROGRAMM, DAS DEN LOKALISIERUNGSMODUS
*                BEI DER EINGABE BENUTZT
*                ------------------------------------
*
CDTE     CSECT
         USING *,11
E10      SAVE   (14,12)
         LR     11,15
         LA     14,RETTGEB
         ST     14,8(13)
         ST     13,4(14)
         LR     13,14
         OPEN   (KARTEIN,(INPUT),BANDAUS,(OUTPUT))
E11      GET    KARTEIN
         MVC    E51,72(1)
         MVC    E52,0(1)
         PUT    BANDAUS,E50
         B      E11
E20      CLOSE (KARTEIN,,BANDAUS)
         FREEPOOL KARTEIN
         FREEPOOL BANDAUS
         L      13,4(13)
         RETURN (14,12),RC=0
KARTEIN  DCB    DSORG=PS,MACRF=GL,
                DDNAME=SYSIN,EODAD=E20
BANDAUS  DCB    DSORG=PS,MACRF=PM,
                DDNAME=TPUT,RECFM=FB,
                BLKSIZE=320,LRECL=80
E50      DS     OCL80         AUSGABEGEBIET FUER DIE AUSGABEDATEI
E51      DS     CL8
E52      DS     CL72
RETTGEB  DC     18F'0'
         END    E10
```

Abb. 22.5. Beispielprogramm, das den L-Modus für die Eingabe benutzt

Das dritte Beispielprogramm (Abb. 22.6.) benutzt den L-Modus für die Ausgabe. Man beachte bei diesem Beispiel besonders, daß das PUT-Makro bereits vor der Verarbeitung erscheinen muß, damit das Gebiet bereitgestellt werden kann, das den Ausgabesatz aufnehmen soll. Das Programm enthält einen Pseudoabschnitt, durch DSECT eingeleitet, durch den die Struktur des Ausgabesatzes beschrieben wird. Für den Pseudoabschnitt stellen wir als Basisregister das Register 1 zur Verfügung (siehe die entsprechende USING-Vereinbarung). Bei dieser einfachen Verarbeitung hätten wir uns natürlich leicht auch ohne diesen Pseudoabschnitt behelfen können. So aber wird uns einmal gezeigt, welche Vorkehrungen in einem Programm getroffen werden müssen, wenn man Pseudoabschnitte verwenden will.

```
*              PROGRAMM, DAS DEN LOKALISIERUNGSMODUS
*              BEI DER AUSGABE BENUTZT
*              ------------------------------------
*
CDTE     CSECT
         USING *,11
E10      SAVE  (14,12)
         LR    11,15
         LA    14,RETTGEB
         ST    14,8(13)
         ST    13,4(14)
         LR    13,14
         OPEN  (KARTEIN,(INPUT),BANDAUS,(OUTPUT))
E11      GET   KARTEIN,E31
         PUT   BANDAUS
         USING CDTF,1
         MVC   E51,E31+72
         MVC   E52,E31
         B     E11
E20      CLOSE (KARTEIN,,BANDAUS)
         FREEPOOL KARTEIN
         FREEPOOL BANDAUS
         L     13,4(13)
         RETURN (14,12),RC=0
KARTEIN  DCB   DSORG=PS,MACRF=GM,                              1
               DDNAME=SYSIN,EODAD=E20
BANDAUS  DCB   DSORG=PS,MACRF=PL,                              1
               DDNAME=TPUT,RECFM=FB,                           2
               BLKSIZE=320,LRECL=80
CDTF     DSECT
E50      DS    0CL80
E51      DS    CL8
E52      DS    CL72
CDTE     CSECT
E31      DS    CL80          ARBEITSGEBIET FUER DIE EINGABEDATEI
RETTGEB  DC    18F'0'
         END   E10
```

Abb. 22.6. Beispielprogramm, das den L-Modus für die Ausgabe benutzt

Das letzte Programm (Abb. 22.7.) zeigt uns typische Verhältnisse in großen, umfangreichen Programmen. Wir verwenden den L-Modus sowohl für die Eingabe als auch für die Ausgabe. Als Basisregister für die Eingabe verwenden wir das Register 2 und als Basisregister für die Ausgabe das Register 1.

Die Instruktion

```
LR    2,1
```

muß dem PUT-Makro vorausgehen, da dieses die vorhergehenden Inhalte der Register 0 und 1 zerstört.

Für die Ausgabedatei wird ein Pseudoabschnitt (DSECT) benutzt, für die Eingabe nicht. In vielen einfachen Fällen, das zeigt dieses Beispiel deutlich, ist es besser, wenn man sich nicht mit Pseudoabschnitten herumquält.

In Ermangelung gegenteiliger Ansichten kann man festhalten, daß es einfacher ist, den M-Modus zu benutzen. Soll die Verarbeitung in einem EA-Gebiet erledigt werden, so ist es sicher einfacher, wenn man dazu die Eingabedatei und nicht die Ausgabedatei heranzieht.

```
*               PROGRAMM, DAS DEN LOKALISIERUNGSMODUS
*               SOWOHL BEI DER EINGABE ALS AUCH BEI DER AUSGABE BENUTZT
*               ------------------------------------------------------
*
CDTE     CSECT
         USING *,11
E10      SAVE  (14,12)
         LR    11,15
         LA    14,RETTGEB
         ST    14,8(13)
         ST    13,4(14)
         LR    13,14
         OPEN  (KARTEIN,(INPUT),BANDAUS,(OUTPUT))
E11      GET   KARTEIN
         LR    2,1
         PUT   BANDAUS
         USING CDTF,1
         MVC   E51,72(2)
         MVC   E52,0(2)
         B     E11
E20      CLOSE (KARTEIN,,BANDAUS)
         FREEPOOL KARTEIN
         FREEPOOL BANDAUS
         L     13,4(13)
         RETURN (14,12),RC=0
KARTEIN  DCB   DSORG=PS,MACRF=GL,                                    1
               DDNAME=SYSIN,EODAD=E20
BANDAUS  DCB   DSORG=PS,MACRF=PL,                                    1
               DDNAME=TPUT,RECFM=FB,                                 2
               BLKSIZE=320,LRECL=80
RETTGEB  DC    18F'0'
CDTF     DSECT
E50      DS    0CL90
E51      DS    CL8
E52      DS    CL72
         END   E10
```

Abb. 22.7. Beispielprogramm, das den L-Modus sowohl für die Eingabe als auch für die Ausgabe benutzt

Aufgabengruppe 60

1) Programmiere das Beispiel von Abschnitt 20.4. neu und benutze dabei das Makro DCB sowie die EA-Makros!

2) Löse die Aufgabe 2) der Aufgabengruppe 56 neu und benutze dabei das Makro DCB!

3) Löse die Aufgabe 3) der Aufgabengruppe 57 für das Betriebssystem OS!

4) Löse die Aufgabe 4) der Aufgabengruppe 57 für das Betriebssystem OS!

5) Löse die Aufgabe 5) der Aufgabengruppe 57 für das Betriebssystem OS!

6) Welche Ausgabe erfolgt durch das nachstehende Programm, wenn auf den Spalten 1 bis 5 die folgenden Eingabewerte vorliegen?

a) 00000 b) 00123 c) 0012K d) 99999

```
BA        CSECT
R         BALR  11,0
          USING *,11
          SAVE  (14,12)
          LR    11,15
          LA    14,RETTGEB
          ST    14,8(13)
          ST    13,4(14)
          LR    13,14
          OPEN  (XIN,INPUT,YOUT,OUTPUT)
S         GET   XIN
          PACK  X,0(5,1)
          ZAP   Y,EINS
          AP    Y,X
          AP    Y,X
          UNPK  YU,Y
          PRTOV YOUT,12
          PUT   YOUT,YU
          B     S
E         CLOSE (XIN,,YOUT)
          FREEPOOL XIN
          FREEPOOL YOUT
          L     13,4(13)
          RETURN (14,12),RC=0
XIN       DCB   DSORG=PS,MACRF=(GL),                    1
                DDNAME=SYSIN,EODAD=E
YOUT      DCB   DSORG=PS,MACRF=(PM),                    1
                DEVD=PR,BUFL=7,PRTSP=1,                 2
                RECFM=F,DDNAME=SYSPRINT,                3
                BLKSIZE=7,BUFNO=2
YU        DS    CL7
X         DS    CL3
Y         DS    CL4
EINS      DC    PL1'-1'
RETTGEB   DC    18F'0'
          END   R
```

7) Ändere die Ausgabe des unter 6) vorgestellten Programmes so ab, daß zuerst der Wert von X aufgelistet wird. Anschließend sollen zwei Leerstellen erscheinen und danach erst der Wert von Y.

8) Die Lösung von Aufgabe 7) ist so abzuändern, daß führende Nullen unterdrückt werden und daß bei negativen Werten das Minuszeichen unmittelbar vor der ersten gültigen Ziffer erscheint.

9) Löse die Aufgabe 9 der Aufgabengruppe 57 für das Betriebssystem OS!

10) Schreibe ein Programm, das den Inhalt von Lochkarten als 80-stellige Sätze einliest. Die eingelesenen Sätze sollen anschließend ungeändert auf ein Magnetband aufgezeichnet werden. Dabei soll keine Blockung der Sätze vorgenommen werden. Es sind Standardkennsätze zu verwenden!

11) Schreibe ein Programm, mit dessen Hilfe das in Aufgabe 10) erstellte Magnetband gelesen werden kann. Die gelesenen Sätze sollen anschließend sofort auf einen Plattenspeicher geschrieben werden. Die daraus entstehende Plattenspeicherdatei soll starr fortlaufend organisiert sein.

12) Die Aufgaben 10) und 11) sollen so abgeändert werden, daß bei allen auf Magnetbändern gespeicherten Dateien der Blockungsfaktor 4 und bei der auf dem Plattenspeicher gespeicherten Datei der Blockungsfaktor 3 Verwendung findet.

13) Die Aufgabenstellung von Aufgabe 11 ist so abzuändern, daß eine logisch fortlaufend organisierte Datei erstellt wird. Der Inhalt der Stellen 3 bis 5 der Sätze soll hierbei als Schlüssel dienen.

22.8. Die Steuerblöcke DECB, ECB und IOB

Bevor wir mit unserer Besprechung fortfahren, wollen wir drei Steuerblöcke kennenlernen. Der Ereignissteuerblock (Event Control Block, abgekürzt ECB) belegt ein Vollwort. Das erste Byte desselben kann die in Abb. 22.8. dargestellten Werte annehmen.

Inhalt des ECB (sedezimal)	*Bedeutung*
80	Das Kanalprogramm ist nach Aufsetzen eines WAIT- oder CHECK-Makros noch nicht abgeschlossen
7F	Fehlerlose Beendigung des Kanalprogrammes
41	Fehlerhafte Beendigung des Kanalprogrammes
42	Abbruch des Kanalprogrammes wegen Verletzung eines SEDZ-Bereiches
44	Kein Starten des Kanalprogrammes, da ein Fehler bei der Festhaltung des Geräteendes entdeckt wurde; erneute Aufsetzung des Kanalprogrammes möglich
48	Das Kanalprogramm ist beseitigt worden
4B	Während der Fehlerbeseitigung bei Magnetbändern war entweder die Adresse des den Fehler verursachenden Kanalbefehlswortes gleich Null oder es wurde unerwartet ein Ladepunkt festgestellt
4F	Während der Fehlerbeseitigung bei einer SEDZ war entweder die Spuradresse oder der Satzbereich R_0 nicht lesbar

Anmerkung: Die restlichen Bytes des Ereignisblockes (ECB) enthalten auf ihren Bitstellen gewöhnlich lauter Nullen

Abb. 22.8. Wertevorrat des ersten Bytes des Ereignissteuerblockes (ECB)

Der Eingabe/Ausgabe-Block, abgekürzt EAB[1], beginnt an einer Vollwortgrenze. Bei Speichergeräten mit direkter Zugriffsmöglichkeit ist er 40 Bytes lang, bei allen anderen Geräten 33 Bytes. Eine Übersicht über seinen Inhalt vermittelt die Abb. 22.9.

Bytes	Inhalt
0	Kennzeichen Bitstelle 0—1 bei Datenkettung Bitstelle 1—1 bei Befehlskettung Bitstelle 6—1 bei einem Kanalprogramm, das keine Bezugnahme zu anderen Kanalprogrammen hat
1	vom System verwaltete Kennzeichen
2 bis 3	Inhalt der ersten beiden Prüfbytes bei Auftreten eines Gerätefehlers
4	Erstes Byte des für dieses Kanalprogramm benutzten Ereignissteuerblockes (ECB)
5 bis 7	Adresse des ECB
8 9 bis 15	vom System verwaltete Kennzeichen Letzte sieben Bytes des Kanalstatuswortes (CSW), gespeichert bei Auftreten der Kanalendebedingung
16	Instruktionslängenschlüssel, Bedingungsschlüssel und Programm-Maske, gespeichert bei Ausführung der Instruktion SIO
17 bis 19	Adresse des ersten Kanalbefehlswortes (CCW)
20	Reserviert für die Benutzung durch das Betriebssystem OS
21 bis 23	Adresse des Steuerblockes (DCB)
24	Reserviert für die Benutzung durch das OS
25 bis 27	Adresse des Kanalbefehlswortes (CCW) bei der Fehlerbeseitigung
28 bis 29	Zahl, um die der Blockzähler jedesmal dann erhöht wird, wenn ein Kanalprogramm ausgeführt ist; normalerweise gleich 1 beim Vorwärtslesen und 1 beim Rückwärtslesen; gilt nur für Magnetbänder. Muß bei anderen Geräten gleich Ø sein
30 bis 31	Zahl der Wiederholungsversuche, die bei der Fehlerbeseitigung aufgesetzt werden
32 bis 39	SEDZ-Adresse (MBBZZKKS)

Abb. 22.9. Aufbau des Eingabe/Ausgabe-Blockes (IOB)

Der Datenereignissteuerblock[2] wird für jede Operation benötigt, die für eine Basiszugriffsmethode aufgesetzt ist. Sein Inhalt ist in Abb. 22.10. dargestellt.

1) Im angloamerikanischen Sprachraum als „Input Output Block", abgekürzt IOB, bezeichnet
2) Im angloamerikanischen Sprachraum als „Data Event Control Block", abgekürzt DECB, bezeichnet

Bytes	Inhalt
0 bis 3	Ereignissteuerblock (ECB)
4 bis 5	Kennzeichen, die den Zugriffstyp betreffen
6 bis 7	Länge des gelesenen oder geschriebenen Blockes
8 bis 11	Adresse des Datensteuerblockes (DCB)
12 bis 15	Adresse des EA-Gebietes
16 bis 19	a) bei BSAM und BDAM: Adresse des EA-Blockes b) bei BISAM: Adresse des logischen Satzes
20 bis 23	bei BIDAM und BDAM: Adresse des Schlüssels
24 bis 25	bei BISAM: Code für Annahmebedingungen
24 bis 27	bei BDAM: Blockadresse
28 bis 31	Nächste Adresse

Abb. 22.10. Aufbau des Datenereignissteuerblockes (DECB)

Ein für die Zugriffsmethode BSAM kreierter Datenereignissteuerblock besitzt eine Länge von 20 Bytes, ein solcher für BISAM 26 Bytes und einer für BDAM 32 Bytes. Es sei uns der Hinweis gestattet, daß der Ereignissteuerblock (ECB) Teil des DECB ist. Der gesamte DECB wird generiert, wenn das READ- bzw. WRITE-Makro expandiert wird. Da das Feld für die logische Satzlänge (LRECL) bei den Basiszugriffsmethoden nicht vermutet wird, muß die Länge eines Satzes undefinierter Satzlänge durch eine bestimmte Codierung ermittelt werden. In Abb. 22.11. ist eine solche Möglichkeit dargestellt.

```
CHECK decb          WARTEN AUF VOLLENDUNG
LH    x,decb+6      LAENGE IM CCW
L     y,decb+16     ADRESSE DES IOB
SH    x,14(y)       LAENGE IM CSW
```

Abb. 22.11. Satzlängenbestimmung (Beispiel)

Die Eingabe/Ausgabe-Blöcke werden bei der Eröffnung der Datenmenge aufgebaut. Bei BSAM ist durch die im NCP-Operanden (siehe Abb. 22.1.) festgelegte Anzahl die Zahl der EAB, die beim Eröffnen konstruiert wird, bestimmt. Aus jedem READ- bzw. WRITE-Makro wird ein eigener DECB geschaffen. Für jeden DECB im Programm muß ein eindeutiger Name vorliegen.

Es ist im Programm möglich, den Ereignissteuerblock zu testen, ob eine Operation vollendet ist oder nicht. Dabei muß der ECB vor Beginn der Operation gelöscht werden. Ist nun das erste Byte des ECB ungleich Null, so ist die Operation abgeschlossen. Der Programmierer darf während einer Operation den Inhalt dieses Bytes nicht verändern, er kann jedoch seinen Inhalt prüfen.

22.9. Puffer für die Basis-Zugriffsmethoden

Die Basis-Zugriffsmethoden sind in erster Linie für den wahlweisen direkten Zugriff zu bestimmten Daten entworfen worden und nicht für den fortlaufenden Zugriff. Deshalb können die Basis-Zugriffsmethoden auch nicht einen Satz im Voraus heranholen, weil nicht erwartet werden kann, daß im gleichen Augenblick vom Programm her eine Anfrage nach den Daten dieses Satzes hinausgeschickt wird.

Es ist weiterhin möglich, daß das Programm zunächst zum Satz S_1 zugreift und anschließend den Satz S_2 heranholt, diesen aber schon längst verarbeitet hat, bevor die Verarbeitung von S_1 beendet ist. Somit muß für die Basis-Zugriffsmethoden nicht nur ein gewisser Vorrat von Puffern in irgendeiner Art von Pufferpool bereitstehen, sondern es muß darüberhinaus die tatsächliche Zuordnung eines Puffers aus dem Pool und seine nachfolgende Freigabe vollständig unter Steuerung des Programmes erfolgen können.

Die normale Verarbeitung, die sich der Unterstützung der Basis-Zugriffsmethoden BSAM, BISAM oder BDAM bedienen muß, verwendet die Makros OPEN und CLOSE genau so wie bei ihrem Einsatz mit den erweiterten Zugriffsmethoden. EA-Operationen werden durch die Makros READ und WRITE eingeleitet. Auf die Vollendung einer durch READ oder WRITE gestarteten Operation kann gewartet werden, wenn man das Makro CHECK codiert. Die Makros BUILD, GETPOOL und FREEPOOL werden ebenfalls so benutzt, wie wir es für die erweiterten Zugriffsmethoden kennengelernt haben. Durch die genannten Makros BUILD, GETPOOL, OPEN, CLOSE und FREEPOOL werden Pufferpools konstruiert und wieder freigegeben. Wenn nun für eine Eingabeoperation ein Puffer benötigt wird, so ist das Makro

```
GETBUF dcb,(r)      bzw.
GETBUF (1),(r)
```

zu codieren. Durch die Ausführung dieses Makros wird ein Puffer aus dem Pool geholt und seine Adresse ins angegebene Register r gestellt. Die Rückgabe dieses Puffers an den Pool erfolgt durch Ausführung des Makros

```
FREEDBUF decb,K,deb      bzw.
FREEDBUF decb,D,dcb
```

Den Datenereignissteuerblock decb können wir auch durch eine Registernotierung, (r) oder (0), ansprechen, wenn wir zuvor seine Adresse in das entsprechende Register laden. Auch für den Datensteuerbock dcb kann eine Registernotierung, (1) oder (r), erfolgen. Durch D ist BDAM gemeint, K verweist auf BISAM. Für BSAM ist eine dynamische Pufferung nicht unterstützt.

Für den Datensteuerblock muß eine Differenz genannt werden. Bei BSAM gilt als Standard für den Operanden BUFNO (siehe Abb. 22.1.) der Wert 0. Das bedeutet, daß kein Pufferpool existiert. In diesem Fall muß der Benutzer selbst Puffer bereitstellen.

22.10. Die Zugriffsmethode BSAM

BSAM kann zur Verarbeitung starr fortlaufend gespeicherter Dateien herangezogen werden. Diese Zugriffsmethode kann aber auch zur Erstellung von Dateien dienen, zu denen später mit Hilfe von BDAM zugegriffen werden soll.

Im OPEN-Makro können für einen DCB die folgenden Angaben aufgeführt werden:

```
INPUT
OUTPUT
INOUT
OUTIN
RDBACK    (nur bei Magnetbandgeräten möglich)
UPDAT     (nur bei SEDZ)
```

Wenn INPUT angegeben ist, so dürfen im Programm keine WRITE-Makros für die betreffende Datei stehen, bei OUTPUT sind keine READ-Makros erlaubt. INOUT bedeutet, daß die Datei zunächst mit INPUT eröffnet wird und später als Ausgabedatei verwendet werden darf, ohne daß sie zwischendurch abgeschlossen und neu eröffnet zu werden braucht. In ähnlicher Weise wird durch OUTIN eine Datei spezifiziert, die als Ausgabedatei eröffnet wird und später ohne Neueröffnung als Eingabedatei verwendet werden kann. Mit RDBACK ist eine Eingabedatei gemeint, die von einem Magnetband rückwärts gelesen werden soll. Durch UPDAT wird eine auf einer SEDZ liegende Datei gekennzeichnet, deren Sätze gelesen und bei Bedarf mit Hilfe des WRITE-Makros durch neue ersetzt werden können. Wird eine Datei mit UPDAT eröffnet, so bedeutet die Niederschrift des WRITE-Makros, daß ein Satz durch einen anderen ersetzt wird (im allgemeinen wird dadurch der gelesene Satz auf den neuesten Stand gebracht); in allen anderen Fällen erzeugt die Codierung von WRITE einen neuen Satz.

Im DD-Statement kann

```
LABEL=(,,,OUT)
```

angegeben werden, wodurch verhindert wird, daß eine Datenmenge mit der Spezifikation OUTIN für Eingabezwecke benutzt wird; in gleicher Weise kann durch

```
LABEL=(,,,IN)
```

die Benutzung einer Datenmenge mit der Spezifikation INOUT für Ausgabezwecke verhindert werden.

Im DD-Statement kann der Operand OPTCD auf die gleichen Optionen wie bei QSAM verweisen. Das gleiche gilt für den Operanden RECFM mit der Ausnahme, daß

```
RECFM=VBST[A|M]
```

nicht erlaubt ist.

Der Operand MACRF darf die folgenden Parameter besitzen:

```
R,RC,RP,W,WC,WP,WL,(R[C],W[C]),(R[P],W[P])
```

Hierbei haben die einzelnen Buchstaben folgende Bedeutungen:

R	Das Makro READ wird verwendet.
W	Das Makro WRITE wird verwendet.
C	Das Makro CNTRL wird verwendet.
P	Das Makro POINT oder das Makro NOTE wird verwendet.
L	Das Makro WRITE wird im Lademodus verwendet.

Die Makros CNTRL, PRTOV und FEOV können wie bei QSAM eingesetzt werden.

Eine Eingabeoperation wird durch das Makro READ gestartet. Seine Form lautet:

```
READ     decb,typ,dcb,gebiet,länge
```

Hierbei stellt decb die Adresse eines Datenereignis-Steuerblockes (Data Event Control Block, kurz auch DECB) dar. Unter typ kann entweder SF für Vorwärtslesen oder SB

für Rückwärtslesen geschrieben werden; die letzte Eintragung ist jedoch nur für Magnetbänder möglich. Unter dcb ist wie üblich die Adresse des DCB festzuhalten. Die Adresse des Eingabepuffers (evtl. durch GETBUF bekommen) ist unter gebiet zu codieren. Unter länge ist die Anzahl von Bytes anzugeben, die bei Sätzen des Formats U gelesen werden sollen. Bei Sätzen im F- oder im V-Format bleibt die Längeneintragung unberücksichtigt und braucht deshalb auch nicht niedergeschrieben zu werden. Wird bei Sätzen, die im U-Format vorliegen, als Längeneintragung 'S' angegeben, so wird der ganze Satz gelesen. Wenn man als Typoperand SB codiert, so muß als Gebietsadresse die Adresse des am weitesten rechts gelegenen Bytes des Puffers aufgeführt werden, also die höchste Adresse eines zum Puffer gehörenden Bytes. Jeder Operand, der eine Adresse oder eine Länge spezifiziert, kann durch eine Registernotierung ersetzt werden. d.h. durch (r). Als Register können hierfür die Register 2 bis 12 gewählt werden, wobei in diese zuvor die entsprechenden Informationen hineingestellt werden müssen. Meist wird der Operand gebiet in dieser Form niedergeschrieben. Betrachten wir ein Beispiel:

```
READ  DECB2,SF,DCB5,(4)
```

Durch diese Makroinstruktion wird das Vorwärtslesen von einer Datei eingeleitet, deren Datensteuerblock mit DCB2 bezeichnet ist; die einzelnen Bytes werden dabei in ein Gebiet gelesen, dessen Adresse im Register 4 enthalten ist. Der Datenereignis-Steuerblock wird hierbei unter der Adresse DECB2 gefunden. Durch GETBUF kann zuvor der Puffer angefordert werden.

Durch das READ-Makro wird die Eingabeoperation nur eingeleitet, anschließend wird die Ablaufsteuerung sofort an das Programm zurückgegeben. Das Makro

```
CHECK decb [,DSORG=IS|ALL]
```

bewirkt, daß die weitere Verarbeitung im Programm solange aufgeschoben wird, bis die Vollendung der Operation durch den entsprechenden Vermerk im ECB angezeigt ist. Da der Ereignissteuerblock bekanntlich Teil des DECB ist, muß auf diesen durch decb bezug genommen werden, was auch durch eine Registernotierung, (r) oder (1), geschehen kann. Wird das Schlüsselwort DSORG benutzt, so bedeutet DSORG=IS, daß das CHECK-Makro sich auf eine Routine bezieht, die nur für die Zugriffsmethode BISAM Gültigkeit besitzt. DSORG=ALL hingegen sorgt dafür, daß eine Routine generiert wird, die gleichermaßen für BSAM, BISAM und BDAM geeignet ist. Das Weglassen von DSORG führt zu einer CHECK-Routine, die nur für BSAM oder BDAM geeignet ist.

Das Makro

```
WAIT  [anzahl],ECB=adresse        bzw.
WAIT  [anzahl],ECBLIST=adresse
```

hebt die Verarbeitung solange auf, bis der Ereignissteuerblock unter adresse die Information enthält, daß die EA abgeschlossen ist. In der ersten Form muß unter anzahl der Wert 1 notiert sein, oder anzahl ist wegzulassen. In der zweiten Form stellt adresse ein Vollwort dar, das die Adresse eines ECB enthält. An dieses Wort schließen sich Wörter mit den Adressen von weiteren Ereignissteuerblöcken an. Bei der letzten Adresse ist die Bitstelle 0 auf 1 gesetzt. Der Programmablauf wird solange aufgeschoben, bis die unter anzahl genannte Zahl von Ereignissteuerblöcken Informationen enthält, die auf die Vollendung der zu gehören EA-Operationen schließen lassen. WAIT unterscheidet sich von CHECK dahingehend, daß immer die nächstfolgende Instruktion ausgeführt wird; bei CHECK können aber ebenso auch die für SYNAD oder EODAD genannten Routinen angesprungen werden.

Das Makro WRITE besitzt die Form:

```
WRITE decb,SF,dcb,gebiet,länge
```

Die Operanden sind die gleichen wie bei dem Makro READ. Wenn die Datei mit UPDAT eröffnet wurde, so wird der Satz bei einer SEDZ zurückgeschrieben. Ist die Datei dagegen mit OUTPUT, OUTIN oder INOUT eröffnet worden, so wird ein neuer Satz in die betreffende Datei gestellt. Wird der Inhalt eines Puffers nicht mehr länger benötigt, so muß nach der Ausführung eines CHECK-Makros ein FREEBUF - Makro hinausgeschickt werden.

Bei anderen Satzformaten außer U sollte der Längenoperand weggelassen werden. Durch WRITE wird die Ausgabe eingeleitet; CHECK oder WAIT sind vonnöten, wenn es gilt, ihre Vollendung zu überprüfen bzw. zu sichern. Ist die Datei mit UPDAT eröffnet worden, so wird durch WRITE derjenige Satz ersetzt, der zuvor, vom gleichen DECB gesteuert, gelesen wurde (wir haben nicht besprochen, wie das zu geschehen hat). In allen anderen Fällen wird durch WRITE ein neuer Satz in die Datei gestellt.

Durch BSAM kann auch eine Datei erstellt werden, zu der später direkt zugegriffen werden soll, d.h. eine regional organisierte Datei. Man verwendet dazu das Makro WRITE in der Form

```
WRITE decb,typ,dcb,gebiet,länge[,folge]
```

Im MACRF-Operanden des DCB-Makros muß dann allerdings der Parameter L angegeben sein (L-Modus). Die Operanden sind die gleichen wie bei jedem anderen WRITE-Makro von BSAM. Eine Ausnahme bildet hier lediglich der Operand typ, für den die folgenden Eintragungen vorgenommen werden können:

SF	Neuer Datensatz
SD	Scheinsatz (Pseudosatz)
SZ	Spurbeschreibungssatz R_0
SFR	Spannsatz (mit BFTEK=R und RECFM=VS)

Unter gebiet muß ein Gebiet genannt sein, daß groß genug ist, um sowohl den Schlüssel als auch die Daten aufnehmen zu können. Eine Ausnahme macht jedoch SZ: Der Operand gebiet kann hier fehlen. Bei SD wird nur ein Datenbyte benötigt. Bei Scheinsätzen ist für das erste Schlüsselbyte der Inhalt FF (sedezimal) vorzusehen. Der letzte Operand folge wird nur zusammen mit SFR benötigt; er gibt ein Gebiet an, in das vom System die nächste relative Spuradresse der nächsten Spur gestellt wird.

Das Makro WRITE stellt ins Register 15 die folgenden sedezimalen Codes hinein:

0	SZ: Eine weitere Spur ist verfügbar; R_0 wurde geschrieben.
	SF, SD: Der Block wurde geschrieben; auf der Spur ist noch weiterer Platz vorhanden.
4	Satzformat F: Der Block wurde geschrieben; die Spur ist anschließend voll.
	Satzformat S, V, U: Es war nicht mehr genügend Platz auf der Spur vorhanden, um den Block zu schreiben; auf die nächste Spur muß R_0 geschrieben werden, bevor der Block aufgezeichnet werden kann.
8	Satzformat F: Der Block wurde geschrieben; der nächste Block erfordert einen anderen Bereich
	Bei Angabe von SZ wurde R_0 geschrieben; der nächste Block erfordert einen weiteren Bereich
C	Der Block wurde nicht geschrieben; es ist CHECK hinauszuschicken, danach erneut WRITE.

Abb. 22.12. zeigt ein Programm, mit dessen Hilfe eine Datei angelegt wird, die später durch die Zugriffsmethode BDAM erschlossen werden kann. Die Notwendigkeit zur Benutzung von BSAM für die Erstellung solcher Datenmengen ist darin zu suchen, daß mit BDAM keine Spurbeschreibungssätze geschrieben werden können. Es sei ausdrücklich betont, daß dieses Programm nur einen Plattenbereich aufbaut. Mit SAVEAREA ist das Rettungsgebiet bezeichnet worden.

```
DAMI      CSECT
          USING *,11
          SAVE  (14,12)
          LR    11,15
          LA    14,SAVEAREA
          ST    14,8(13)
          ST    13,4(14)
          LR    13,14
          OPEN  (I,OUTPUT)
          LA    8,8
DAMIW     WRITE DECBW,SD,I,J
          CR    8,15
          BE    DAMIZ
          CHECK DECBW
          B     DAMIW
DAMIZ     CHECK DECBW
          CLOSE (I)
          L     13,4(13)
          RETURN (14,12),RC=0
I         DCB   DSORG=PS,MACRF=WL,                1
                BLKSIZE=43,KEYLEN=5,              2
                LRECL=43,RECFM=F,                 3
                DDNAME=I
SAVEAREA  DC    18F'0'
J         DC    CL6' '
          END   DAMI
```

Anmerkung: Folgezeilen sind durch eine Notierung auf der Spalte 43 kenntlich gemacht.

Abb. 22.12. Erstellung einer Datei mit Hilfe von BSAM

Das Makro

```
NOTE  dcb     bzw.
NOTE  (1)     bzw.
NOTE  (r)
```

stellt in das Register 1 oder in das Register r die Kennzeichnung eines Blockes. Bei Bändern ist das die duale Reihenfolgenummer des letzten gelesenen oder geschriebenen Blockes. Der erste Block auf einem Band trägt stets die sedezimale Nummer 00000001. Bei Plattenspeichern wird in die vier Bytes des Registers 1 oder r eine Zahl in der Form TTSz hineingestellt. Hierbei gibt TT die Spurnummer relativ zur ersten Spur der Datei an (die erste Spur ist durch 00 gekennzeichnet), S ist die Satznummer auf der Spur (der erste Datensatz auf einer Spur ist durch 1 gekennzeichnet) und z ist gleich 00 (sed).

Wenn die letzte Operation durch ein WRITE-Makro hervorgerufen wurde, so enthält bei Plattenspeichern das Register 0 außerdem zzLL, wobei durch LL die Anzahl der auf der Spur verbleibenden Bytes festgehalten ist (LL wird in sedezimaler Form angegeben). Wie bei Register 1 bedeutet z den Wert 00 (sed).

Das Makro POINT besitzt folgende Form:

```
POINT dcb,platz
```

Durch dieses Makro kann die normale Verarbeitungsreihenfolge geändert werden, indem die Kennzeichnung des nächsten zu lesenden bzw. zu schreibenden Satzes festgelegt wird. Die zweite Eintragung bezieht sich auf ein volles Wort, in dem der Blockzähler bei Magnetbändern und TTRz bei SEDZ stehen muß. Durch z wird wiederum 00 (sed) angezeigt. CC gibt die laufende Blocknummer in einer Datei an; TT gibt die relative Spurnummer an (bezogen auf den Dateianfang) und R ist die Blocknummer innerhalb der Spur (der erste Block hat die Nummer 1).

Der erste Block in jeder Datei kann durch 0000 0001 angesprochen werden. Bei Plattenspeichern wird durch die Kennzeichnung TTR1, deren letztes Byte also 01 (sed) enthält, der unmittelbar auf den Block mit der Kennzeichnung TTRz folgende Block angesprochen. Eine solche Bezugnahme ermöglicht es also, daß der nächste Satz auf Bändern wie auch auf Plattenspeichern einfach dadurch angesprochen werden kann, indem eine 1 auf die vorhergehende Kennzeichnung addiert wird.

Der erste Operand beim POINT-Makro kann selbstverständlich auch in Registernotierung, (1) oder (r), vermerkt werden, wenn zuvor in das betreffende Register die Adresse des Datensteuerblockes (DCB) geladen wird.

22.11. Die Zugriffsmethode BISAM

Die Zugriffsmethode BISAM wird durch DSORG=IS im DCB-Statement selektiert. Die Parameter von MACRF können wie folgt lauten:

```
(R[S][C])
(WA[C])
(WU[A][C])
(R[S][C],WA[C])
(RU[S][C],WU[A][C])
```

Die Verwendung der in den eckigen Klammern stehenden Kennbuchstaben ist wie üblich freigestellt. Im übrigen kommt den einzelnen Buchstaben die folgende Bedeutung zu:

R	Verwendung des Makros READ
W	Verwendung des Makros WRITE
C	Verwendung des Makros CHECK (andernfalls muß WAIT benutzt werden)
A	Hinzufügen neuer Sätze zur Datenmenge
S	Vorkehrungen für dynamische Pufferzuordnung
U	Vorkehrungen für die Aufdatierung von Sätzen

Die Makros OPEN, CLOSE, BUILD, GETPOOL, FREEPOOL, GETBUF, FREEBUF, CHECK und WAIT werden wie bei der Zugriffsmethode BSAM benutzt. Eine Ausnahme ist jedoch zu beachten: Alle Optionen, die im OPEN-Statement erscheinen, werden ignoriert.

Das Makro

```
READ  decb,K[U],dcb,gebiet,länge,schlüssel
```

ist ähnlich dem gleichen Makro von BSAM. Durch K ist das normale Lesen festgelegt; KU dagegen gibt an, daß der gelesene Satz wieder zurückgeschrieben werden soll (UPDAT). Wenn für den Operanden gebiet nur 'S' niedergeschrieben wird, dann stellt das Steuerprogramm automatisch einen Puffer für den zu lesenden Satz bereit. Die Adresse dieses

Gebietes findet man im DECB. Der Operand länge gibt die maximale Länge des zu lesenden Satzes an. Wenn diese Länge kleiner als die physikalische Satzlänge ist, so wird der letzte Teil des Satzes nicht in den Hauptspeicher eingelesen. Die Eintragung 'S' bewirkt, daß der gesamte Satz gelesen wird. Bei Dateien, die aus geblockten Sätzen bestehen, ist 'S' die einzige korrekte Eintragung. Der letzte Operand schlüssel ist die Adresse des Feldes, das den Satzschlüssel des Satzes enthält, zu dem zugegriffen werden soll. Registernotierungen können für bestimmte Operanden vorgenommen werden, z.B. (r) für dcb, gebiet, länge und schlüssel. Bei der dynamischen Pufferung (Angabe von 'S' für gebiet) ist für die Freigabe des automatisch vom Betriebssystem zugeordneten Puffers das Makro FREEDBUF zu benutzen.

Das Makro WRITE besitzt folgende Form:

```
WRITE decb,typ,dcb,gebiet,länge,schlüssel
```

Die Operanden haben mit einer Ausnahme die gleiche Bedeutung wie beim READ-Makro. Für typ ist KN aufzuführen, wenn ein neuer Satz geschrieben, K dagegen, wenn ein geänderter Satz zurückgeschrieben werden soll. Ein WRITE-Statement mit der Typangabe K muß sich auf den Datenereignissteuerblock (DECB) beziehen, mittels dessen ein READ-Statement mit der Typangabe KV aufgesetzt war. Kein Satz kann hinzugefügt werden, wenn nicht zuvor alle zwecks Aufdatierung gelesenen Sätze, die andere DECB benutzen, zurückgeschrieben sind.

Dem EA-Gebiet müssen 16 Bytes vorausgehen, die das System für seine Zwecke benötigt. Der zu schreibende Satz darf erst auf die nachfolgenden Stellen plaziert werden. Bei vorgesehener dynamischer Pufferung kann auf ein READ mit KV ein WRITE mit K folgen, wobei 'S' spezifiziert werden kann. In diesem Fall wird der Puffer automatisch nach der Beendigung des Schreibens freigegeben; das Makro FREEDBUF wird hierbei nicht benötigt.

Die Länge wird durch 'S' bezeichnet, es sei denn, daß ein Satz variabler Länge aufdatiert wird, wobei sich seine Länge geändert hat.

Die Makros WAIT oder CHECK müssen eingesetzt werden, wenn es gilt, auf die Vollendung der Operation zu warten. Beide speichern bei auftretenden Ausnahmebedingungen bestimmte Bits in den DECB (siehe Abb. 22.13.).

READ

Bitstelle	Bedeutung
0	Kein Satz gefunden
1	Falsche Satzlänge
4	Nicht korrigierbarer EA-Fehler
5	Nicht erreichbarer Block
6	Überlaufsatz

WRITE

Bitstelle	Bedeutung
0	Kein Satz gefunden (nur beim Zurückschreiben)
1	Falsche Satzlänge
2	Kein Platz gefunden, um einen Satz hinzuzufügen (nur für das Hinzufügen aktuell)
3	Der beim Lesen verwendete Puffer wurde freigegeben, bevor das Zurückschreiben erfolgen konnte (gilt nur für das Zurückschreiben)
4	Nicht korrigierbarer EA-Fehler
5	Nicht erreichbarer Block
7	Hinzuzufügender Satz bereits vorhanden (gilt nur beim Hinzufügen)

Abb. 22.13. Ausnahmebedingungen beim Lesen und Schreiben

22.12. Die Zugriffsmethode BDAM

Dateien, zu denen mit BDAM zugegriffen werden soll, müssen mit INPUT, OUTPUT oder UPDAT eröffnet werden. Wenn sie mit INPUT eröffnet werden, so können keine WRITE - Makros für sie gegeben werden. Wenn sie dagegen mit OUTPUT oder UPDAT eröffnet werden, so sind alle Makros, einschließlich des READ-Makros, erlaubt. Es besteht kein Unterschied ob man eine Datei mit OUTPUT oder mit UPDAT eröffnet.

Dem Operanden MACRF des DCB-Makros können die folgenden Parameter zugewiesen sein:

```
(RI[S][C])
(RK[I][S][C])
(WI[C])
(WK[I][C])
(WA[K][I][C])
(R[K][I][S][C],W[A][K][I][C])
```

Bei der letzten Form muß R entweder von K oder I gefolgt werden; auf A muß einer der Kennbuchstaben A, K oder I stehen. Die einzelnen Kennbuchstaben haben folgende Bedeutungen:

R	Das Makro READ wird verwendet
W	Das Makro WRITE wird verwendet
C	Das Makro CHECK wird verwendet
K	Die Sätze werden durch ihren Schlüssel aufgesucht
I	Die Sätze werden durch ihre Kennzeichnung aufgesucht
A	Es werden Sätze zur Datei hinzugefügt
S	Es ist dynamische Pufferung vorgesehen

Der Operand RECFM des DCB-Statements kann wie folgt angegeben werden:

```
RECFM=F[T]
RECFM=V[[B]S]
RECFM=U
```

Hierbei bedeutet T, daß die Spurenüberlaufeinrichtung (Spurenverbindung) benutzt werden soll.

Bei BDAM ist der Operand OPTCD im DCB-Statement äußerst wichtig. Er kann die folgenden Parameter annehmen:

```
[R|A][E][F][W]
```

Die einzelnen Buchstaben haben folgende Bedeutungen:

W	Gültigkeitsprüfung beim Schreiben
E	Erweitertes Suchen, wie durch LIMCT spezifiziert. Es ist dabei nur READ mit DK und WRITE mit DA erlaubt. Außerdem wird E ignoriert, wenn A aufgeführt ist
A	Bei READ und WRITE ist die Blockadresse auf der SEDZ durch MBBZZKKS spezifiziert, d.h. durch die aktuelle Blockadresse
B	Bei READ und WRITE ist BBB spezifiziert, d.h. die relative Blockadresse ist angegeben
F	Es ist eine Rückmeldung in der Form vorgesehen, wie sie bei den READ- und WRITE-Makros angegeben ist. Fehlt F, so wird die Form MBBZZKKS benutzt

Ist weder A noch R angegeben, so beziehen sich die Makros READ und WRITE auf TTS, d.h. auf die relative Spuradresse des Blockes.

In BDAM kann also ein Satz auf drei verschiedenen Wegen angesprochen werden:

MBBZZKKS	Echte Adresse
TTS	Relative Spuradresse
BBB	Relative Satzsadresse (nur bei RECFM=F)

Bei Verwendung echter Adressen bedeutet M die Bereichsnummer auf dem Datenträger. Die weitere Buchstabenfolge BBZZKKS legt Datenzelle (B), Zylinder (Z), Kopf oder Spur (K) und Satznummer (S) fest. Die Benutzung echter Adressen bewirkt, daß die Datei unbeweglich (nicht verschieblich) ist (DSORG=DAU). Bei relativen Spuradressen gibt TT die relative Spurnummer und S die Satznummer auf der betreffenden Spur an. Die erste Spur einer Datei hat dabei die relative Spurnummer TT=00; der erste Satz einer Datei ist somit durch TTS=001 ausgewiesen. Relative Satzadressen sind nur bei Sätzen der Satzart F erlaubt; BBB ist hierbei die laufende Nummer des Satzes innerhalb der Datei. Die niedrigste relative Satznummer ist 000.

Beim Lesen und beim Schreiben muß der Programmierer die Spur und den Satz auf der Spur angeben. Bei Verwendung echter Adressen bzw. bei Verwendung relativer Spuradessen kann er nur die ersten sieben Bytes der echten Adresse bzw. die ersten beiden Bytes der relativen Spuradresse angeben; darauf muß allerdings der Schlüssel folgen, nach dem die Datei durchsucht werden soll. Der Programmierer kann aber auch festlegen, daß das S bei MBBZZKKS bzw. bei TTS benutzt wird; in diesem Fall braucht kein Schlüssel angegeben zu werden. Bei relativen Satzadressen wird niemals ein Schlüssel zum Durchsuchen der Datei eingesetzt.

Das Makro READ tritt in der folgenden Form auf:

```
READ   decb,typ,dcb,gebiet,länge,schlüssel,stz,folge
```

Hierbei bedeuten:

decb	Adresse des DECB
dcb	Adresse des DCB
gebiet	Adresse des Eingabepuffers
schlüssel	Adresse des Feldes, das den Schlüssel enthält (vorausgesetzt, es wird ein Schlüssel verwendet)
stz	Adresse des Feldes, das die Suchadresse enthält

Unter typ können folgende Kennungen erscheinen:

```
DI[F][R[U]]
DK[F][R[U]]
```

Durch DI ist festgelegt, daß die Daten und (falls vorhanden) der Schlüssel von der unter stz aufgeführten SEDZ-Adresse zu lesen sind. DK hingegen sagt aus, daß die Daten, jedoch kein Schlüssel, von der unter stz genannten Spur zu lesen sind, wobei anschließend die Spur nach einem Block durchsucht wird, diesen Schlüssel unter schlüssel zu finden ist. Durch F wird Rückmeldung gefordert, d.h. daß am Schluß der Operationen die externe Speicheradresse des Blockes nach stz gestellt wird. R sagt aus, daß unter folge die Rückmeldung der Adresse des nächstfolgenden Datensatzes erfolgt; nur für Spannsätze ist ein solches Vorgehen möglich. V verweist auf ein modifiziertes Vorgehen gegenüber R: die Rückmeldung betrifft den nächsten Datensatz oder den Spurbeschreibungssatz R_0, je nachdem, welcher zuerst kommt. Der Operand folge ist bei Codierung von R wegzulassen.

Für dcb kann auch die Registernotierung r erfolgen. Für gebiet kann die Adresse des Eingabepuffers, (r) oder 'S' geschrieben werden; 'S' verweist auf die Benutzung der dynamischen Pufferung. Unter länge kann die tatsächliche Länge durchgeführt werden; wird (r) codiert, so muß das genannte Register die Länge enthalten. Auch die Angabe von 'S' ist möglich; greift man auf diese Form zurück, so wird die unter BLKSIZE genannte maximale Anzahl von Bytes gelesen. Für schlüssel kann eine Adresse, aber auch (r), 0 oder 'S' geschrieben werden; hierdurch wird die Adresse des Schlüssels festgehalten, wenn das Suchen über den Schlüssel erfolgt; im anderen Fall wird der Schlüssel in dieses Feld eingelesen. Soll der Schlüssel nicht gelesen werden, so ist 0 zu codieren. Wird für gebiet 'S' codiert, so ist 'S' auch für schlüssel erlaubt; es hat zur Folge, daß der Schlüssel in den Puffer gelesen wird und dort unmittelbar vor den Daten gespeichert wird.

Unter stz ist ein Feld zu nennen, das entweder drei Bytes oder acht Bytes lang ist; es kann auch durch eine Registernotierung spezifiziert werden. Bei OPTCD=F ersetzt MBBZZKKS, TTS oder BBB den Inhalt dieses Feldes, wenn die Leseoperation abgeschlossen ist. Ist unter OPTCD jedoch F nicht angegeben, so wird MBBZZKKS in dieses Feld gestellt, wenn der Kennbuchstabe F Teil des Operanden typ ist. Unter folge ist TTS für den nächsten Satz zu finden, wenn R für Sätze des Formats VS oder VBS angeben ist.

Das Makro WRITE besitzt die Form:

```
WRITE decb,typ,dcb,gebiet,länge,schlüssel,stz
```

Die Operanden sind mit Ausnahmen die gleichen wie beim Makro READ. Unter typ können folgende Kennbuchstaben angegeben werden:

```
DA[F]
DI[F]
DK[F]
```

F bezieht sich, wie vom READ-Makro her bekannt, auf Rückmeldungen. Durch DI wird ein Satz auf die Stelle geschrieben, die unter stz genannt ist. Wird DA angegeben, so werden Daten geschrieben, wobei nur der unter schlüssel genannte Schlüssel zur Lokalisierung des Satzes benutzt wird. Das Durchsuchen beginnt auf der unter stz festgehaltenen Spur und ist durch die Angabe bei LIMCT begrenzt. Bei Angabe von DA wird ein neuer Satz auf den ersten verfügbaren Platz geschrieben, der unter stz addressiert und durch LIMCT begrenzt ist. Der Operand gebiet kann zusätzlich zu (r) oder 'S' codiert werden; 'S' ist nur dann möglich, wenn sich das vorangegangene READ-Makro, das zu dem fraglichen Satz zugegriffen hat, auf die dynamische Pufferung stützte. In diesem Fall muß die im Datenereignissteuerblock (DECB) des READ-Makros stehende Gebietsadresse in das entsprechende Feld des DECB des WRITE-Makros hineingebracht werden. Die Angabe von 'S' bewirkt dann auch die Pufferfreigabe. Der Operand länge kann in den Formen (r), 'S' oder als absoluter Ausdruck gebraucht werden. Die Angabe von 'S' hat zur Folge, daß der Wert von BLKSIZE herangezogen wird. Eine solche Operandenform ist nur bei Sätzen des U-Formats brauchbar. Unter schlüssel ist die Adresse des Schlüsselfeldes aufzuführen, und zwar in der Form 0 (bei Nichtverwendung eines Schlüssels) oder (r) oder 'S'. Die letzte Angabe ist nur erlaubt, wenn 'S' auch für den gleichen Operanden im READ-Makro niedergeschrieben wurde.

Eines der Makros CHECK oder WAIT muß benutzt werden, um auf die Vollendung der Operation zu warten. Die Makros OPEN, CLOSE, BUILD, GETBUF, FREEBUF und FREEPOOL sind identisch mit denen von BSAM. Das CLOSE-Statement mit T ist allerdings für BDAM nicht erlaubt.

Im Anschluß an das CHECK- oder das WAIT-Makro kann der Inhalt der beiden mittleren Bytes des Ereignissteuerblockes ECB überprüft werden. In diesen (siehe Abb. 22.14.) sind Ausnahmebedingungen vermerkt, die beim Lesen oder auch Schreiben aufgetreten sind.

Byte 1

Bitstelle	Bedeutung
0	Satz nicht innerhalb der vorgesehenen Durchsuchungsgrenzen gefunden
1	Die angegebene Länge ist nicht gleich der echten Länge
2	Innerhalb der Durchsuchungsgrenzen kein Platz vorhanden, um einen Satz hinzuzufügen (gilt nur für WRITE)
3	Nicht alle Bitstellen von Byte 2 enthalten 0
4	Nicht korrigierbarer EA-Fehler
5	Dateiendesatz (EOF) mit der Datenlänge 0
6	Nicht korrigierbarer darauf bezogener Fehler
7	Vom OS reserviert

Byte 2

Bitstelle	Bedeutung
0	Nicht benutzt
1	WRITE-Makro; Datei jedoch mit INPUT eröffnet
2	LIMCT gleich 0, jedoch erweitertes Suchen angefordert
3	Suchadresse nicht innerhalb der Grenzen der Datei
4	Verletzung des Schutzes eines Spurbeschreibungssatzes (nur bei WRITE möglich)
5	(a) Es wurde ein Schlüssel angegeben, obwohl im DCB-Makro keine vorgehen waren (b) Es ist keine Schlüsseladresse vorhanden
6	Der Typoperand des READ- oder des WRITE-Makros steht im Widerspruch mit dem OPTCD-Operanden des DCB
7	Es wurde der Versuch unternommen, einen neuen Satz hinzuzufügen, dessen Schlüssel mit FF (sed) beginnt (nur bei RECFM=F und WRITE)

Abb. 22.14. Ausnahmebedingungen bei EA-Operationen

22.13. Listen- und Ausführungsformen von Makros

Die Auflösung der READ- und WRITE-Makros schließt zweierlei ein:

1) Erzeugung eines Datenereignissteuerblockes (DECB),
2) Generierung von Instruktionen zum Aufruf der geeigneten IOCS-Routine

Die beiden Makros

```
READ  decb,typ,[dcb],[gebiet],[länge],[schlüssel],[stz],[folge],MF=L
WRITE decb,typ,[dcb],[gebiet],[länge],[schlüssel],[stz],[folge],MF=L
```

dienen allein zur Generierung eines DECB. Der erste Operand muß dabei der DECB-Name sein; der zweite Operand muß vorhanden sein. Alle anderen Operanden mit Ausnahme des letzten brauchen nur bei Bedarf hinzugestellt zu werden. Normalerweise werden jedoch alle angegeben, vorausgesetzt, man benötigt sie in der Form, die für den Anwendungsfall ausreicht. Für keinen Operanden ist die Registernotierung erlaubt. Der letzte Operand lautet:

```
MF=L
```

wodurch die Listenform des Makros gekennzeichnet ist. Instruktionen werden bei Vorliegen dieser Form nicht generiert, nur eben, wie gesagt, der DECB, READ und WRITE können ohne Probleme miteinander ausgetauscht werden.

Zusammen mit den Listenformen müssen die Ausführungsformen der Makros benutzt werden. Sie lauten:

```
READ  decb,typ,[dcb],[gebiet],[länge],[schlüssel],[stz],[folge],MF=E
WRITE decb,typ,[dcb],[gebiet],[länge],[schlüssel],[stz],[folge],MF=E
```

Durch diese Makroformen wird kein DECB generiert; es werden allerdings Instruktionen erzeugt, die im DECB die Inhalte jener Felder ersetzen, die durch die Operanden angesprochen werden. Hinzu kommen die Instruktionen zum Aufruf der IOCS-Routinen. Der erste Operand ist der DECB-Name. Der zweite Operand muß unbedingt angegeben sein. Alle anderen Operanden mit Ausnahme des letzten brauchen nur dann aufgeführt zu werden, wenn es notwendig ist, die bereits im DECB stehenden Werte durch andere zu ersetzen. Der letzte Operand muß

```
MF=E
```

lauten; durch ihn ist ausgesagt, daß es sich um die Ausführungsform des Makros handelt. Mit Ausnahme von typ und MF=E können alle anderen Operanden eine Registerbezugnahme in der Form (r) darstellen.

Die Ausführungsform muß benutzt werden, wenn es gilt, Sätze aufzudatieren. Listen- und Ausführungsformen können in allen Programmen benutzt werden; sie können durchaus auch vermischt mit den Standardformen der Makros auftreten. Nach wie vor muß freilich darauf geachtet werden, daß für jede simultan ablaufende EA – Operation ein DECB zur Verfügung stehen muß.

22.14. Das physische IOCS

Zum Zwecke der physischen EA sind im DCB-Statement die Operanden DDNAME, EXLST, EODAD, DEVD, KEYLEN, DEN, STACK und PRTSP genau so zu benutzen wie bei jeder anderen Datei auch (siehe Abb. 22.1.). Ebenso sind erlaubt:

```
MODE={E}
     {C}

DSORG={PS}
      {DA}
      {IS}

OPTCD=Z
```

Der letzte Operand bezeichnet die eingeschränkte Bandfehlerbeseitigung. Notwendig ist die Benutzung der in Abb. 22.15. beschriebenen Operanden.

Vom Benutzer muß ein eigener EA-Block (IOB)[1] und ein eigener Ereignissteuerblock (ECB) eingerichtet werden. Außerdem ist ein Kanalprogramm zu schreiben. In OPEN-Makro sind die Optionen INPUT, RDBACK und OUTPUT erlaubt sowie REREAD, LEAVE und DISP. Die Angabe von INPUT bewirkt die Prüfung von Nachsätzen. OUTPUT hat die Aufzeichnung von Nachsätzen zur Folge. Im CLOSE-Makro dürfen LEAVE, REREAD und DISP aufgeführt sein. Die Angabe von T ist nicht erlaubt. Außerdem kann das Makro

```
EOV   dcb
```

1) Im angloamerikanischen Sprachgebrauch als "Input-Output-Block", abgekürzt IOB, bezeichnet.

Operand	Bedeutung
IOBAD=name	Name des vom Benutzer eingerichteten Eingabe/Ausgabe-Blockes (IOB)
IMSK=wert	Behandlung der EA-Fehler und ihre Beseitigung; alle von der IBM unterstützten Fehlerroutinen werden nicht benutzt, unter wert kann eine beliebige Bezugnahme erfolgen
MACRF=(E)	Es wird eine konsekutiv organisierte, auf einem Datenträger residierende Datei verarbeitet
MACRF=({R / W})	R und W haben die gleiche Bedeutung: der Datenträgerwechsel wird vom Betriebssystem unterstützt; eingeschlossen sind dabei das Schreiben und/oder das Prüfen von Plattenkennsätzen.

Abb. 22.15. Notwendige Operanden des DCB-Statements für die physische EA

benutzt werden; es hat den Datenträgerwechsel zur Folge. Der automatische Übergang zum nächsten Datenträger bei Feststellung des Endes eines Datenträgers wird nicht unterstützt. Es ist Sache des Programmierers, selbst ein EOV zu codieren. Der Operand dcb kann auch in Registernotierungen (1) oder (r) erscheinen.

Die Ausführung eines Kanalprogrammes wird durch das Makro

```
EXCP   eab
```

eingeleitet. Der Operand eab stellt die Adresse des EA-Blockes (IOB) dar; im Bedarfsfall kann dafür auch (1) oder (r) geschrieben werden. Auf die Vollendung der Operation muß gewartet werden; hierfür ist das Makro WAIT zu benutzen. Danach sind die einzelnen Bestandteile des IOB zu prüfen. Der Ereignissteuerblock wird bei der Kanalendebedingung versorgt. Soll jedoch das Geräteende-Bedingung maßgebend für die Versorgung des ECB sein, so ist ein Kanalbefehlswort (CCW), das keine Operation beinhaltet, an das Kanalprogramm anzuketten. Fehler, die vor dem Auftreten der Kanalendebedingung entdeckt werden, bewirken, daß das System einen Prüfbefehl hinausschickt und die ersten beiden Prüfbytes in den IOB stellt. Fehler, die nach dem Kanalende auftauchen, bewirken, daß X'44 000 000' oder, bei einem verwandten Kanalprogramm X'48 000 000' in den ECB gestellt werden, wenn das nächste EXCP-Makro zum Tragen kommt. Geschieht das, so kann das EXCP-Makro wiederholt werden.

In der Abb. 22.16. ist ein Programm dargestellt, das Lochkarten mit dem physischen IOCS liest und die in ihnen enthaltenen Informationen dazu benutzt, die Datenmenge, die durch das Programm von Abb. 22.12. erstellt wurde, aufzudatieren. Als Zugriffsmethode dient dabei BDAM. Auf den Lochkarten sind folgende Daten untergebracht:

Spalten 1 bis 35: Artikelbezeichnung (NDESCR)
Spalten 36 bis 44: Artikelnummer (NPART)
Spalten 45 bis 51: Einzelpreis (NCOST)
Spalten 52 bis 58: Lagerbestand (NQTY)
Spalten 59 bis 79: nicht benutzt
Spalte 80 : Kennzeichnung (NCODE)

1, 3, 5, 7 — Neue Artikelbezeichnung	Aufdatierung eines existierenden Satzes
2, 3, 6, 7 — Neuer Einzelpreis	
4, 5, 6, 7 — Neuer Lagerbestand	
8 — Neuer Artikel (Hinzufügung eines neuen Satzes)	

Die Plattensätze haben folgendes Format:

a) Schlüssel
   Stellen 1 bis 5: Artikelnummer (DPART), gepackte Dezimaldarstellung

b) Daten
   Stellen 1 bis 35: Artikelbezeichnung (DDESCR)
   Stellen 36 bis 39: Einzelpreis (DCOST), gepackte Dezimaldarstellung
   Stellen 40 bis 43: Lagerbestand (DQTY), gepackte Dezimaldarstellung

Die relative Spurnummer (TTS) wird aus dem Quotienten von der Artikelnummer und der Zahl 8 388 609 errechnet. Der Quotient liegt zwischen 0 und 119; wir benutzen ihn als relative Spurnummer. Die Plattendatei belegt somit 120 Spuren. Würden wir das erweiterte Suchen vorsehen, so erfordert das die Bereitstellung zusätzlicher Spuren, um Überläufe aus den letzten Spuren unterbringen zu können.

Das Programm liefert einen Rückkehrcode von 0 ab, wenn alle vorgesehenen Änderungen verarbeitet sind. In anderen Fällen kommt es zum Rückkehrcode 8.

Das Makro SNAP mit dem dazugehörigen DCB wird im nächsten Abschnitt behandelt. Es berührt in keiner Weise die Programmlogik.

```
         ISEQ  76,80
EXDA     CSECT
         USING *,11
         SAVE  (14,12)
         LR    11,15
         LA    14,SAVEAREA
         ST    14,8(13)
         ST    13,4(14)
         LR    13,14
         OPEN  (I,UPDAT,X,INPUT)
         OPEN  (SNAP,OUTPUT)
         SR    10,10
EXDA2    EXCP  Y
         WAIT  ECB=V
         TM    Y+12,X'01'   EOF
         BO    EXDA9
         CLI   NCODE,C'1'
         BL    EXDA8
         CLI   NCODE,C'8'
         BH    EXDA8
         PACK  DPART,NPART
         ZAP   WORK,DPART
         DP    WORK,PRIME
         CVB   0,WORK+5
         STH   0,T
         READ  B,DK,MF=E
         WAIT  ECB=B
         SNAP  DCB=SNAP,PDATA=REGS,
               STORAGE=(B,B+3,D,D+47)
         CLI   NCODE,C'8'
         BNE   EXDA3
         TM    B+1,X'80'
         BZ    EXDA8
         TM    B+1,X'7F'
         BNZ   EXDA8
```

Abb. 22.16. (Teil 1). Beispielprogramm (1. Teil)

```
         MVC   DDESCR,NDESCR
         PACK  DCOST,NCOST
         PACK  DQTY,NQTY
         WRITE B,DA,MF=E
         B     EXDA7
EXDA3    TM    B+1,X'FF'
         BNZ   EXDA8
         TM    NCODE,X'01'
         BNO   EXDA4
         MVC   DDESCR,NDESCR
EXDA4    TM    NCODE,X'02'
         BNO   EXDA5
         PACK  DCOST,NCOST
EXDA5    TM    NCODE,X'04'
         BNO   EXDA6
         PACK  DQTY,NQTY
EXDA6    WRITE B,DK,MF=E
EXDA7    WAIT  ECB=B
         SNAP  DCB=SNAP,STORAGE=(B,B+3)
         TM    B+1,X'FF'
         BZ    EXDA2
EXDA8    LA    10,8
         B     EXDA2
EXDA9    CLOSE (I,,X)
         CLOSE (SNAP)
         LR    15,10
         L     13,4(13)
         RETURN (14,12),RC=(15)
SNAP     DCB   DSORG=PS,BLKSIZE=882,          1
               RECFM=VBA,MACRF=(W),           2
               LRECL=125,DDNAME=SNAP
I        DCB   DSORG=DA,BLKSIZE=43,           1
               RECFM=F,MACRF=(RK,WAK),        2
               KEYLEN=5,DDNAME=I
X        DCB   DSORG=PS,EODAD=EXDA9,          1
               MACRF=(E),DDNAME=X,            2
               DEVD=RD,IOBAD=Y
Y        DS    0D              IOB
         DC    5X'00'
         DC    AL3(V)
         DC    9X'00'
         DC    AL3(W)
         DC    A(X)
         DC    8X'00'
V        DC    F'0'            ECB
         READ  B,DK,I,DD,'S',D,T,MF=L
W        CCW   X'02',N,X'00',80
         DS    0D
N        DS    0CL80           CARD
NDESCR   DS    CL35
NPART    DS    ZL9
NCOST    DS    ZL7
NQTY     DS    ZL7
NFILLER  DS    CL21
NCODE    DS    CL1
D        DS    0CL48           DISK
DPART    DS    PL5             KEY
```

Abb. 22.16. (Teil 2). Beispielprogramm (2. Teil)

```
DD        DS    OCL43          DATA
DDESCR    DS    CL35
DCOST     DS    PL4
DQTY      DS    PL4
SAVEAREA  DC    18F'0'
T         DC    XL3'0'         TTR
PRIME     DC    PL8'19'
WORK      DS    PL13
          END   EXDA
```

Anmerkung: Folgezeilen sind durch ein vom Leerzeichen verschiedenes Zeichen auf der Spalte 43 angekündigt.

Abb. 22.16. (Teil 3). Beispielprogramm (3. Teil)

22.15. Makros für Speicherauszüge

Das Makro

```
ABEND wert [,DUMP]      bzw.
ABEND (1)[,DUMP]
```

bewirkt die anomale Beendigung eines Jobsteps. Dabei kann unter wert ein Vollendungsschlüssel angegeben werden; er sollte ein vielfaches von 4 und kleiner als 4 096 sein. Dieser Wert kann in den EXEC-Steuerstatements der Jobsteuersprachen für nachfolgende Jobsteps getestet werden. Wird DUMP niedergeschrieben, so wird der Inhalt des Arbeitsspeichers (Hauptspeichers) in Sedezimalziffern und in Zeichen herausgegeben (gedruckt), und zwar in die Datei, für die ein DD-Statement mit den Dateinamen SYSABEND oder SYSUDUMP existiert. In SYSUDUMP wird nur der Abzug der eigenen Partition hineingestellt, in SYSABEND zusätzlich auch der Abzug des Supervisorgebietes. Ein typisches DD-Statement könnte wie folgt aussehen:

```
//SYSUDUMP   DD   SYSOUT=A
```

Fehlt die Angabe von DUMP, so erfolgt kein Speicherabzug. Ist kein DD-Statement für SYSUDUMP oder SYSABEND vorhanden, so erfolgt kein Abzug. Ein verkürzter Auszug der Register etc. wird jedoch als Teil der Auflistung der Jobsteuersprache heraus gegeben. Der erste Operand des ABEND-Makros kann auch durch (r) ausgedrückt werden, wobei r eines der Register 2 bis 12 dargestellt.

Ein nützliches Werkzeug für die Fehlersuche ist das Makro

```
SNAP  DCB=dcb
      [,ID=nummer]
      [,PDATA=(feld,···)]
      [,STORAGE=(anfang,ende,···)]
      [,LIST=adresse]
```

Durch nummer wird eine Zahl zwischen 1 und 127 festgelegt, die als Indentifikation für die Druckausgabe dient. Im Operanden PDATA können Bezugnahmen auf folgende Felder vorliegen:

a) PSW
   Programmstatuswort
b) REGS
   Register

c) SA
Rettungsgebiete
d) IPA
Gesamter virtueller Speicher des Jobsteps
e) ALL
Alle unter a) bis d) genannten Felder

Wird im Operanden nur ein Feld genannt, so können die umgebenden Klammern entfallen. Unter STORAGE können ein oder mehrere Adreßpaare genannt sein. Die erste Adresse stellt den Beginn, die zweite das Ende eines virtuellen Speichergebietes dar, dessen Inhalt in sedezimaler Form und in Zeichenform gedruckt werden soll. LIST bezieht sich auf eine Adreßliste mit Anfangs- und Endadressen. Die linksbündige Bitstelle der letzten Adresse einer solchen Liste muß dabei auf 1 gesetzt sein. STORAGE und LIST können nicht zusammen angegeben sein. Unter DCB muß auch der Name der Datei genannt sein, die die Ausgabe aufnehmen soll. Für den DCB müssen die folgenden Erklärungen vorliegen:

```
dcb     DCB   DSORG=PS,RECFM=VBA,BLKSIZE=882,
              MACRF=(W),LRECL=125,DDNAME=name
```

Unter name kann mit Ausnahme von SYSABEND und SYSUDUMP jeder beliebige Name aufgeführt sein. Vor Ausführung eines SNAP-Makros muß das OPEN-Statement für die betreffende Datei ausgeführt sein. Ebenso sollte das Abschließen vor dem Programmende (CLOSE) erfolgen. Alle Operandenwerte[1] können auch in Registernotierung vorliegen.

Abb. 22.16. zeigt die Benutzung des SNAP-Makros zum Drucken des ECB-Inhalts für die Plattendatei. Durch das erste SNAP-Statement wird auch der Inhalt des Eingabegebietes derPlattendatei ausgegeben. Da SNAP-Makros für ihre Ausführung Zeit benötigen, sollen sie entfernt werden, wenn die Fehlersuche bei einem Programm als abgeschlossen gelten kann.

Aufgabengruppe 61

1) Löse die Aufgabe 5) der Aufgabengruppe 58 unter Verwendung der Zugriffsmethode QSAM zum Beschreiben des Magnetbandes.

2) Löse die Aufgabe 5) der Aufgabengruppe 58 unter Verwendung der Zugriffsmethode BSAM zum Schreiben des Magnetbandes.

3) Ändere die Aufgabe 2) so ab, daß an Stelle eines Magnetbandes ein Plattenspeicher angesprochen wird.

4) Ändere die Aufgabe 2) so ab, daß eine logisch fortlaufend organisierte Datei angesprochen wird und bringe die in Frage kommenden Sätze auf den neuesten Stand.

5) Löse die Aufgabe 9) der Aufgabengruppe 59 so, daß die Zugriffsmethode QSAM zum Beschreiben des Magnetbandes eingesetzt wird.

6) Löse die Aufgabe 10) der Aufgabengruppe 59!

7) Löse die Aufgabe 13) der Aufgabengruppe 59. Zur Erstellung der Datei auf dem Plattenspeicher soll die Zugriffsmethode BSAM eingesetzt werden.

8) Löse die Aufgabe 14) der Aufgabengruppe 59; verwende dabei die Zugriffsmethode BDAM zum Aussuchen der zu verarbeitenden Sätze.

1) Beim Betriebssystem OS, VS 2, kann unter BLKSIZE auch der Wert 1632 vorliegen

9) Ändere das Programm von Abb. 22.12. wie folgt ab: Die Datei belegt mehr als drei Bereiche und ist zu initialisieren.

10) Ändere das Programm von Abb. 22.16. wie folgt ab:

a) Zum Lesen der Karten ist die Zugriffsmethode BSAM einzusetzen.
b) Die Plattendatei ist indexsequentiell organisiert (Zugriffsmethode ISAM).

22.16. Supervisor-Makroinstruktionen

In diesem Abschnitt wollen wir einige Makros kennenlernen, die nicht mit Programmverbindungen in Zusammenhang stehen, sondern die einige Supervisorhilfen anbieten.

Wenn ein GETPOOL-Makro ausgeführt wird, so muß ein bestimmtes Hauptspeichergebiet angefordert und in einen Pufferpool eingeteilt werden. Der Programmierer kann für seine eigenen Zwecke ebenfalls ein bestimmtes Hauptspeichergebiet anfordern, indem er das Makro GETMAIN in einer der folgenden Formen niederschreibt:

```
GETMAIN EC,LV=n,A=addr
GETMAIN EU,LV=n,A=addr
GETMAIN R,LV=n
GETMAIN R,LV=(Ø)
GETMAIN VC,LA=addr1,A=addr2
GETMAIN VU,LA=addr1,A=addr2
```

Die Registernotierung kann für n, addr, $addr_1$ und $addr_2$ benutzt werden. EV und EL als Operanden bewirken die Anforderung eines Einzelgebietes virtuellen Speichers mit der Länge von n Bytes. Die Adresse dieses Gebietes wird vom Supervisor nach addr gestellt. Register 15 wird bei erfolgreicher Anforderung auf 0 gesetzt, anderenfalls auf 4. Bei EC fährt das Programm mit der nächsten Instruktion fort, bei EU kommt es zu einem anormalen Programmabbruch, wenn das angeforderte Gebiet nicht zur Verfügung gestellt werden konnte. R bedeutet das gleiche wie EU, nur wird die Adresse des zugeteilten Gebietes ins Register 1 gestellt. Mittels VU und VC wird ein Gebiet variabler Länge angefordert. LA verweist dabei auf die Adresse eines achtstelligen Gebietes; sein erstes Wort muß die akzeptierbare minimale Gebietslänge, sein zweites die maximale enthalten. Das System stellt das größte Gebiet zur Verfügung, dessen Länge kleiner als oder gleich der maximalen Länge ist. Seine Adresse kann unter $addr_2$ gefunden werden. Register 15 wird auf 0 oder auf 4 gesetzt. Das Programm wird weitergeführt, wenn VC codiert ist. Bei Niederschrift von VU endet das Programm anomal, wenn nicht einmal ein Gebiet minimalen Umfanges bereitgestellt werden kann.

```
BNDRY=PAGE   oder
BNDRY=DBLWD
```

kann zu den E- bzw. V-Formen zusätzlich hinzugefügt werden. Durch ihn wird angezeigt, daß das angeforderte Gebiet auf Seitengrenze (PAGE) oder auf Doppelwortbegrenzung (DBLWD) beginnen soll. Fehlt die Angabe von BNDRY, so wird BNDRY=DBLWD unterstellt. Der Umfang des bereitgestellten Gebietes ist stets ein Vielfaches von 8; gibt der Programmierer einen anderen Wert an, so rundet das System die Gebietslänge auf.

Ein auf diese Weise angefordertes Hauptspeichergebiet kann für den Supervisor wieder freigegeben werden, wenn man das Makro FREEMAIN verwendet. Dieses kann in den folgenden Formen eingesetzt werden:

```
FREEMAIN E,LV=n,A=addr
FREEMAIN V,A=addr1
FREEMAIN R,LV=n,A=addr
FREEMAIN R,LV=n,A=(1)
FREEMAIN R,LV=(Ø),A=addr
```

Durch E wird die Rückgabe eines Gebietes des Umfanges n veranlaßt, dessen Anfangsadresse im durch addr adressierten Wort enthalten ist. V bewirkt die Rückgabe eines Gebietes, dessen Adresse in $addr_1$ enthalten ist; seine Länge wird $addr_1 + 4$ entnommen. Durch R wird ein Gebiet mit n Bytes Umfang zurückgegeben, dessen Adresse im durch addr adressierten Wort steht. Das Programm wird anomal beendet, wenn die Adresse des zurückzugebenden Gebietes nicht durch 8 teilbar ist. Für n, addr und $addr_1$ kann auch die Registernotierung benutzt werden.

Wird R codiert, so wird das Rettungsgebiet, auf das Register 13 zeigt, nicht von den GETMAIN- bzw. FREEMAIN-Makros berührt. Somit sind diese Makros befähigt, daß mit ihrer Hilfe ein Rettungsgebiet erlangt und später auch wieder zurückgegeben werden kann.

Durch das Makro

```
PGRLSE  LA=addr1,HA=addr2
```

wird ein von den Grenzen $addr_1$ und $addr_2$ (obere Grenze) eingeschlossenes virtuelles Speichergebiet festgelegt. Für $addr_1$ kann auch die Registernotierung (0) oder (r) erfolgen, für $addr_2$ kann auch (1) oder (r) geschrieben werden. Beide Adressen müssen durch den Seitenumfang des Systems (2048 bei VS1, 4096 bei VS2) teilbar sein. Die Inhalte der kompletten Seiten von $addr_1$ bis $addr_2$ können eingebüßt werden. Der Speicher bleibt zwar mit dem Programm verhaftet, doch darf der Inhalt verlustig gehen. Durch dieses Makro kann die Systembelastung vermindert werden, in dem die Notwendigkeit, Seiten ein- und auszulagern, deren Inhalte nicht länger benötigt werden, eliminiert wird.

22.17. Makros für den Informationsaustausch mit dem Bedienungspersonal

Im Fall, daß die Notwendigkeit vorliegt, die Maschinenbedienung über ein Geschehnis zu informieren, können wir das Makro

```
WTO   'nachricht'[,ROUCDE=n]                    bzw.
WTO   ('nachricht'[,zeile])[,ROUTCDE=n)]
```

benutzen. Die unter nachricht angegebene Zeichenkette wird als Konstante vom C-Typ ausgegeben. Wenn der Operand ROUTCDE niedergeschrieben ist, so wird die Nachricht auf den Konsolen angezeigt, die durch n identifiziert sind. Der Wert 11 für n ist reserviert für Nachrichten an den Programmierer[1]; derartige Nachrichten erscheinen in der Auflistung der Jobsteuersprachen-Statements des betreffenden Programmes, jedoch nicht über eine Bedienerkonsole. Im übrigen werden die zulässigen Werte für n vom Systemprogrammierer vergeben. Fehlt der Operand ROUTCDE, so geht die Nachricht zur Systembedienerkonsole.

Sind mehrzeilige Nachrichten erforderlich, so ist unter zeile C zu notieren, wenn die erste (oder Kontrollzeile) gemeint ist, L steht für Kennsatzzeilen, D für Detailzeilen, DE für die letzte Detailzeile und E für eine Zeile nach der letzten Detailzeile. Die Angabe

1) Im angloamerikanischen Sprachraum unter „Write to programmer", abgekürzt WTP, bekannt.

von E zeigt an, daß die vorhergehende Zeile die letzte Zeile war; die Nachricht auf einer durch E gekennzeichneten Zeile wird ignoriert. Wenn vorhanden, muß die durch C angekündigte Zeile als erste kommen. Einer L-Zeile muß eine C-Zeile oder eine andere L-Zeile vorausgehen.

Jede Nachricht darf nicht mehr als 121 Zeichen umfassen; eine Ausnahme machen hierbei WTP-Nachrichten, die eine beliebige Länge aufweisen dürfen. Bei mehrzeiligen Nachrichten sind C-Zeilen auf 31 Zeichen[1] und andere Zeilen auf 71 Zeichen[2] beschränkt.

Im Register 1 wird eine Zahl zurückgegeben, die die Nachricht identifiziert (Identifikationsnummer).

Das Makro

```
WTOR  'nachricht',adr,länge,ecb[,ROUTCD=n]
```

veranlaßt, daß zunächst die codierte Nachricht angezigt wird. Danach hat der Bediener Gelegenheit, eine Antwort in der Länge einzutippen, die durch den dritten Operanden festgelegt ist; diese Antwort wird in die Hauptspeicherplätze eingelesen, die mit der im zweiten Operanden angegebenen Adresse beginnen. Der Operand gibt schließlich die Adresse eines Ereignis-Steuerblockes an (ECB). Der Ereignis-Steuerblock muß eine Länge von einem Wort besitzen. Der Programmierer muß der Bitstelle 1 (Vollendungsbit) eine 0 zuweisen, bevor er dieses Makro verwendet. Diese Bitstelle wird auf 1 gesetzt, wenn die Eingabeoperation vollendet ist.

Um den Computer in den Wartestatus zu versetzen, bis die durch den betreffenden Ereignis-Steuerblock spezifizierte Operation abgeschlossen ist, kann das WAIT-Makro verwendet werden. Auch bei diesem Makro steht nach seiner Ausführung eine Identifikationsnummer für die Nachricht im Register 1.

Wird eine Nachricht an eine Bildschirmkonsole geschickt, so bleibt sie dort solange sichtbar, bis sie durch die Ausführung des Makros

```
DOM   MSG=r
```

zerstört wird. Hierbei stellt r eines der Register 1 bis 12 dar; in diesem muß die Identifikationsnummer der zu löschenden Nachricht stehen. Um mehrere verschiedene Nachrichten auf einmal zu zerstören, können wir die Form

```
DOM   MSGLIST=adresse
```

benutzen, wobei adresse auch als Registernotierung (1) oder (r) erscheinen kann. Unter adresse können bis zu 60 Vollwörter, je eine Identifikationsnummer enthaltend, zusammengefaßt sein. Die linksbündige Bitstelle des letzten Wortes muß auf 1 gesetzt sein.

Das Makro

```
POST  ecb[,code]
POST  (1)[,code]
POST  ecb[,(Ø)]
POST  (1)[,(Ø)]
```

bewirkt, daß der aufgeführte Ereignis-Steuerblock (ECB) als vollendet gilt; dabei ist es gleichgültig, ob das zugehörige Ereignis eingetreten ist oder nicht. Der Wert für code wird rechtsbündig in die 30 letzten Bitstellen des ECB hineingestellt. Fehlt der zweite Operand, so wird für code standardmäßig der Wert 0 herangezogen.

22.18. Makros für Zeitbestimmungen

Datum und Tageszeit können durch das Makro

```
TIME  DEC
TIME  BIN
TIME  TU
TIME  MIC,adresse
TIME
```

festgestellt werden. Bei allen Formen steht nach Ausführung dieses Makros im Register 1 der Wert 00 jjttty (jj letzte zwei Ziffern der Jahreszahl, ttt laufender Tag des Jahres, y gleich sedezimal F). Den einzelnen Operanden kommt folgende Bedeutung zu: Wird DEC angegeben, so enthält das Register 0 nach Ausführung dieses Makros die Tageszeit in der Form HHMMSSzh (HH Stunden, MM Minuten, SS Sekunden, z Zehntelsekunden und h Hundertstelsekunden); es wird dabei ein 24-stündiger Tag vorausgesetzt. Der im Register 0 stehende Wert liegt in gepackter Dezimaldarstellung vor. Wenn BIN angegeben wird, so wird im Register 0 die Zeit als Vielfaches von 0,01 Sekunden gespeichert. Bei Angabe von TU enthält das Register 0 die Zeit in Zeiteinheiten; eine Zeiteinheit dauert dabei nahezu 26,04 166 μs. Bei vorliegendem ersten Operanden MIC wird die Tageszeit in einem achtstelligen Gebiet gespeichert, das unter adresse gefunden wird. Die Bitstelle 51 dieses Doppelwortes entspricht dabei einer Mikrosekunde.

Nach Ablauf eines vorgegebenen Zeitintervalles kann eine Unterbrechung durch das Makro

```
STIMER REAL[,adresse],intervall  bzw.
STIMER WAIT,intervall
```

hervorgerufen werden. Wenn WAIT angegeben wird, so geht der Computer bis zum Ablauf des aufgeführten Intervalles in den Wartestatus über. Bei der Eintragung von REAL fährt der Computer mit seiner normalen Arbeitsweise fort und verzweigt zu der bei adresse beginnenden Routine, wenn es zu der Unterbrechung kommt. Wenn keine Adresse aufgeführt ist, so fährt das Programm nach der Unterbrechung mit der nächsten Instruktion fort. Im Makro kann für adresse auch eine Registernotierung, (0) oder (r), geschrieben werden.

Der Operand intervall kann wie folgt niedergeschrieben werden:

a) DINTVL=adr
   Durch adr wird Bezug auf ein Doppelwort genommen, das das Zeitintervall in der Form HHMMSSzh in gezonter Dezimaldarstellung enthalten muß.

b) BINTVL=adr
   Durch adr wird Bezug auf ein Wort genommen, das in binärer Form die Zeit in Vielfachen von 0,01 Sekunden enthalten muß.

c) TUINTVL=adr
   Durch adr wird Bezug auf ein Doppelwort genommen, das die Zeit in Zeiteinheiten enthalten muß (Binärdarstellung).

d) TOD=adr
   Durch adr wird Bezug auf die Tageszeit genommen, zu der das Zeitintervall als beendet betrachtet werden soll. Die Tageszeit muß in der Form HHMMSSzh (gezonte Dezimaldarstellung) in einem Doppelwort stehen.

Der Parameter adr kann auch in Form einer Registernotierung, (1) oder (r), geschrieben werden.

Beim Eintritt in die Zeitunterbrechungsroutine nach Verstreichung des vorgegebenen Zeitintervalles bleiben die Registerinhalte 2 bis 12 ungeändert, das Register 13 enthält die Adresse des vom Supervisor bereitgestellten Rettungsgebietes (Save Area), das Register 14 die Rückkehradresse (wodurch ein Rücksprung zum Supervisor ermöglicht wird) und das Register 15 die Adresse der Unterbrechungsroutine (wodurch dieses Register auch als Basisregister verwendet werden kann). Die Unterbrechungsroutine sollte mit einem Rücksprung nach der im Register 14 stehenden Adresse enden, damit vom Supervisor die Rückkehr zum Programm durchgeführt werden kann.

Das Makro

```
TTIMER [CANCEL]
```

stellt ins Register 0 die verbleibende Zeit eines Intervalles, das durch das Makro STIMER aufgesetzt wurde. Die Zeit ist hierbei stets in Zeiteinheiten angegeben. Wenn CANCEL aufgeführt wird, so wird die restliche Intervallzeit auf 0 gesetzt, gleichgültig, wie groß sie noch war. Wenn die Intervallzeit bereits verstrichen war, so wird im Register 0 der Wert 0 zurückgegeben und der Operand CANCEL bleibt ohne Effekt.

22.19. Makros zur Behandlung von Programmunterbrechungen

Das Makro

SPIE adresse,($\left\{\begin{matrix}x\\(y,z)\end{matrix}\right\}\left[,\left\{\begin{matrix}x\\(y,z)\end{matrix}\right\}\right]\cdots$)

spezifiziert einen Speicherplatz mit der Adresse adresse, dem die Programmablaufsteuerung dann übergeben wird, wenn es zu einer Programmunterbrechung mit dem Code x gekommen ist, bzw. bei Programmunterbrechung mit Codes zwischen y und z. Es dürfen so viele Werte für x bzw. y und z geschrieben werden, wie für erforderlich erachtet wird. Somit bewirkt z.B.

```
SPIE  MARIANNE,((1,3),6,(8,11),4)
```

daß die bei MARIANNE beginnende Routine immer dann angesprungen wird, wenn Unterbrechungen mit den Codes 1, 2, 3, 4, 6, 8, 9, 10 oder 11 auftreten. Die Niederschrift dieses Makros ohne Operanden, also

```
SPIE
```

setzt alle zuvor eingeführten Unterbrechungsroutinen außer Kraft. Die Wirkungssphäre dieses Makros reicht nicht bis zur Hinausschickung des nächsten SPIE-Makrobefehles. Werden die Unterbrechungsschlüssel 8, 10, 13 oder 14 spezifiziert, so werden die entsprechenden Markenbitstellen im PSW so geändert, daß die entsprechenden Unterbrechungen auch durchschlagen. Die restlichen Markenbitstellen sind natürlich auf Null gesetzt. Unterbrechungsschlüssel, die größer als 15 sind, dürfen nicht angegeben werden.

Für den Jobstep gibt es ein Programm-Unterbrechungselement[1]. Byte 0 dieses Elementes wird von Supervisor benutzt. Die Bytes 1 bis 3 enthalten die Adresse eines Steuergebietes für Programmunterbrechungen[2], und zwar desjenigen Gebietes, das für das letzte gegebene SPIE-Makro eingerichtet wurden. In den Bytes 4 bis 11 steht bei der Basissteuerung nach einer Programmunterbrechung das alte Programmstatuswort (PSW). Die Bytes 12 bis 31 werden zum Speichern der Registerinhalte benutzt, was durch eine auf die Register 14 bis 2 bezogene STM-Instruktion veranlaßt wird.

1) Im angloamerikanischen Sprachraum als „Program Interruption Element", agbekürzt PIE bezeichnet.
2) Im angloamerikanischen Sprachraum als „Program Interruption Control Area", abgekürzt PICA bezeichnet.

PICA ist ein Gebiet, das sechs Bytes umfaßt. Das Byte 0 enthält auf den vorderen vier Bitstellen Nullen, auf den hinteren die Programm-Marke. Die Bytes 1 bis 3 enthalten die Adresse der Ausgangsroutine. Die Bytes 4 und 5 nehmen die Unterbrechungsmarke auf.

Das erste SPIE-Makro, das in einem Programm vorkommt, gibt der Wert 0 mit Register 1 zurück. Jedes nachfolgende SPIE-Makro stellt ins Register 1 die Adresse des Steuergebietes (PICA), das zum vorhergehenden SPIE-Makro gehört. Dieses Vorgehen ermöglicht es einer Programmprüfungsroutine, ihr eigenes SPIE-Makro hinauszuschicken und dann später die gesetzten Werte des vorangegangenen SPIE-Makros zurückspeichern. Im besonderen dient dazu die Form

```
SPIE   MF=(E,(1))
SPIE   MF=(E,(r))
```

Im angegebenen Register 1 oder r muß dabei die Adresse des Steuergebietes stehen, das zu dem SPIE-Makro gehört, dessen Wirksamkeit wiederhergestellt werden soll.

Kommt es zu einer Programmunterbrechung und geht dabei die Programmablaufsteuerung zur Ausgangsroutine über, so bleiben die Inhalte der Register 2 bis 13 ungeändert. Das Register 14 enthält die Rückkehradresse des Steuerprogrammes, Register 15 die Adresse der Ausgangsroutine. Der Inhalt des Registers 0 kann nicht vorausgesagt werden, während Register 1 die Adresse des PIE aufnimmt. Wenn die Ausgangsroutine die Ablaufsteuerung zurück an das Steuerprogramm gibt, werden die Register 14 bis 2 vom PIE zurückgeladen; außerdem wird die LPSW-Instruktion zum Laden des PSW benutzt, das im PIE untergebracht ist. In der Ausgangsroutine können die in den Bytes 8 bis 11 des PIE stehende Rückkehradressen sowie beliebige Registerinhalte modifiziert werden. Benutzt die Ausgangsroutine die Register 3 bis 13, so müssen selbstverständlich deren Inhalte weggespeichert und vor der Rückkehr wieder geladen werden. Das Register 13 zeigt auf das Rettungsgebiet des Hauptprogrammes. Dieses Gebiet darf von der Ausgangsroutine nicht benutzt werden.

Die Bytes 6 und 7 des PIE enthalten den Unterbrechungsschlüssel.

Die Simulation von Instruktionen für die Handhabung von Zahlen in erweiterter Gleitpunktdarstellung sowie für die Division von Zahlen in erweiterter Gleitpunktdarstellung erfolgt über eine SPIE-Ausgangsroutine. Das Makro

```
DXR    r1,r2
```

wird zu

```
B2 2D ØØ r1r2
```

aufgelöst. Das führt zu einer Ausnahmebedingung für die Operation. Bei Maschinen ohne Gleitpunktarithmetik erweiterter Genauigkeit kommt es ebenfalls zu einer solchen Ausnahmebedingung. Zwei Simulatoren stehen zur Verfügung, einer für Maschinen, bei denen nur die DXR-Instruktion ersetzt werden muß, und ein zweiter für solche Maschinen, bei denen alle Instruktionen ersetzt werden müssen. Der Modul IEAXPSIM identifiziert den für ein spezielles System benötigte Simulator. Zum Laden des Simulators kann der in Abb. 22.17. dargestellte Code benutzt werden.

Die Eingangsadresse für den Simulator ist unter SIMADD zu finden. Der Simulatoraufruf ist in Abb. 22.18. gezeigt.

Zu Abb. 22.18. ist zu bemerken, daß die Adresse des PIE zuvor nach PARMB gespeichert sein muß. Nach der Rückkehr vom Simulator ist Register 15 auf

Null gesetzt, wenn die Simulation erfolgreich war, andernfalls auf X'ØØØØØØFF'. Der Unterbrechungsschlüssel kann geändert werden, damit bei einem evtl. Auftreten der Codes 1, 4, 5, 6, 12, 13, 14 oder 15 während der Simulation solche Unterbrechungsbedingungen auch genau so durchschlagen, wie wenn sie bei einer normalen Instruktionsausführung aufgetreten wären. Der Bedingungsschlüssel im alten PSW wird ebenfalls geändert, wenn eine AXR- oder eine SCR-Instruktion simuliert wird.

```
         LOAD   EP=IEAXPSIM
         LR     15,Ø
         LA     1,PARMA
         BALR   14,15
         DELETE EP=IEAXPSIM
         LOAD   EXPLOC=SIMUL
         ST     Ø,SIMADD
PARMA    DS     X'8Ø',AL3(SIMUL)
SIMUL    DS     D
SIMADD   DC     F'Ø'
```

Abb. 22.17. Instruktionen zum Laden des Sinmlators

```
         L      15,SIMADD
         LA     1,PARMB
         BALR   14,15
PARMB    DS     F              ADDRESSE DES PIE
         DC     A(SIMSV)
         DC     A(WORK)
         DC     X'8Ø',AL3(ZERO)
ZERO     DC     X'ØØ'
WORK     DS     5ØD
SIMSV    DS     16F
```

Abb. 22.18. Aufruf des Sinmlators

Ein Rettungsgebiet sollte vorgesehen werden, wenn der Simulator aufgerufen wird.

Kann im Programm bestimmt werden, daß der Simulator nicht länger benötigt wird, so kann der von ihm belegte Speicherplatz durch das Makro

```
DELETE EPLOC=SIMUL
```

freigegeben werden.

In den meisten Fällen ist in der Ausgangsroutine vorgesehen, die Inhalte der Register 3 bis 13 in ein Rettungsgebiet wegzuspeichern. Danach wird ein neues Rettungsgebiet angelegt und seine Adresse ins Register 13 gestellt; erst dann wird der Simulator aufgerufen. Nach der Rückkehr wird der Inhalt des Registers 15 getestet. Ist dieser ungleich Null, so behandelt die Routine den Unterbrechungsschlüssel im alten PSW. Der Aufruf des Simulators für eine Instruktion, die nicht die erweiterte Gleitpunktdarstellung betrifft, führt im Simulator zum Unterbrechungsschlüssel 1.

22.20. Einfache Programmverbindungen

Aus dem Makro

```
SAVE   (r1[,r2])[,T]
```

wird eine ST- oder STM-Instruktion erzeugt, die die Inhalte der Register r_1 bis r_2 in ein Standard-Rettungsgebiet weglegt. Fehlt die Angabe von r_2, dann kommt es nur zur Speicherung des Inhaltes von r_1. Wird T niedergeschrieben, so wird zum Retten der Registerinhalte 14 und 15 eine gesonderte Instruktion generiert, es sei denn, für r_1 ist 14, 15, 0, 1 oder 2 genannt. In diesem Fall werden die Inhalte aller Register bis einschließlich r_2 gerettet. Ist T niedergeschrieben und r_2 weggelassen, dann werden nur die Inhalte der Register 14, 15 und r_1 weggespeichert. Gewöhnlich wird das Makro in der Form

```
SAVE   (14,12)
```

benutzt.

Zur Rückspeicherung des Inhalts der geretteten Register sowie zum Rücksprung zur aufrufenden Programmeinheit, können wir das Makro RETURN verwenden, das in den folgenden beiden Formen codiert werden kann:

```
RETURN [(r1[,r2])][,T][,RC=aa]
RETURN [(r1[,r2])][,T][,RC=(15)]
```

Diese Makroinstruktion bewirkt, daß die Inhalte der Register r_1 bis einschließlich r_2 aus dem Rettungsgebiet zurückgespeichert werden, auf den das Register 13 verweist. Wenn T angegeben ist, so wird nach Rückspeicherung der Registerinhalte das linksbündige Byte des Wortes 4 (das den Inhalt des Registers 14 enthält) im Rettungsgebiet auf FF (sed) gesetzt, wodurch angezeigt wird, daß der Rücksprung erfolgt ist. Wenn die Eintragung RC vorgenommen wurde, so wird der absolute Ausdruck aa, der kleiner als 4096 sein muß, ins Register 15 geladen. Das geschieht aber erst, nachdem alle Registerinhalte wieder zurückgespeichert sind. Dieser Rückkehrcode kann durch ein EXEC-Statement getestet werden, wenn das RETURN-Makro gleichzeitig zur Beendigung eines Jobsteps geführt hat.

Das Makro RETURN speichert die Registerinhalte zurück und verzweigt zu der im Register 14 stehenden Adresse. Im OS kann im Register 15 ein Wert zurückgegeben werden, und das Makro RETURN kann am Ende eines Jobs dazu dienen, um die Programmablaufsteuerung an den Supervisor zurückzugeben. Wird dieses Makro in einem Hauptprogramm ausgeführt, so geht die Programmablaufsteuerung zum Jobsteuerungsprogramm zurück, das den Rückkehrcode mit dem im COND-Parameter spezifizierten Wert vergleicht, d.h. mit dem Wert des COND-Parameters in einem geeigneten JOB- oder EXEC-Statement.

Das Makro RETURN wird ebenfalls dazu verwendet, um aus den Routinen zur Behandlung von Unterbrechungen zurückzukehren.

Fehlt die Angabe von r_1 und von r_2, so werden keine Registerinhalte zurückgespeichert. Ein Rückkehrcode kann jedoch nach wie vor spezifiziert werden, und die Instruktion

```
BR     14
```

wird nach wie vor generiert.

Wenn ein Programm in einzelnen Programmeinheiten geschrieben wurde, dann ist es notwendig, die Programmablaufsteuerung von einer Programmeinheit in eine andere zu überführen. Das kann sehr einfach durch Verwendung des Makros CALL geschehen. Dieses Makro kann wie folgt codiert werden:

CALL eingangsname[,(arg$_1$,···,arg$_n$)[,VL]][,ID=aa]

Hierbei ist durch den ersten Operanden der externe Name festgelegt, zu dem die Programmablaufsteuerung übergehen soll. Falls es gewünscht wird, kann als erster Operand auch die Registernotierung (15) niedergeschrieben werden; in diesem Fall muß jedoch der Programmierer vorher die Adresse des Eingangspunktes in das Register 15 laden. Wenn Argumente der aufgerufenen Programmeinheit übergeben werden sollen, so müssen diese in Klammern eingeschlossen und durch Kommas voneinander getrennt werden. Es wird vom Makro eine Liste der Adressen der einzelnen Argumente zusammengestellt, und die Adresse dieser Liste wird vom Makro ins Register 1 geladen. Wenn VL codiert wird, dann wird das Vorzeichenbit der letzten Adresse in der Adreßliste negativ gemacht. Diese Möglichkeit sollte ausgenutzt werden, wenn die Anzahl der Argumente variabel ist. Wenn ID=aa niedergeschrieben wird, dann wird nach der Generierung der Konstanten und Instruktionen, die zum CALL-Makro gehören, eine zusätzliche Nop-Instruktion erzeugt, die den absoluten Ausdruck aa als Dualzahl in ihren letzten beiden Bytes enthält; der Wert von aa darf 65536 nicht überschreiten. Die Nop-Instruktion ist dann normalerweise die erste Instruktion, die nach der Rückkehr von der aufgerufenen Programmeinheit ausgeführt wird.

Die aus dem CALL-Makro erzeugten Instruktionen und Definitionen sind in Abb. 22.19. zusammengestellt.

```
         CNOP   2,4
marke    L      15,=V(eingangsname)
         LA     14,*+6+4*n
         BALR   1,15
         DC     A(arg1,···,argn-1)
         DC     X'8Ø'
         DC     AL3(argn)
         DC     XL4'47ØØeeee'
```

Abb. 22.19. Auflösung des Makros CALL

Hierbei bedeutet eeee den absoluten Ausdruck, der durch ID=aa spezifiziert wurde. Wenn ID nicht angegeben ist, so unterbleibt die Generierung der Nop-Instruktion. Es wäre noch zu bemerken, daß vom aufgerufenen Programm für den Rücksprung die Adresse der Nop-Instruktion benutzt wird, nämlich *+6+4· n.

Normalerweise befindet sich eine aufgerufene Programmeinheit im Hauptspeicher. Ist das jeodch nicht der Fall, so sucht das Steuerprogramm nach ihr und lädt sie, bevor an sie die Programmablaufsteuerung übertragen wird. Damit die Suche erfolgreich gestaltet werden kann, muß die aufgerufene Programmeinheit mit der aufrufenden durch den Linkage Editor verbunden worden sein, wobei der OVLY-Parameter des Linkage Editors benutzt sein muß.

Der Programmierer braucht nicht unbedingt die Makros CALL, SAVE und RETURN zu benutzen, da aus ihnen kein Maschinencode erzeugt wird, der von privilegierten Instruktionen Gebrauch macht. Es ist jedoch wesentlich einfacher, sich an die Makros zu erinnern, als eine äquivalente Instruktionsfolge mit den dazugehörigen Definitionen niederzuschreiben.

22.21. Dynamische Programmverbindungen

Um ein Lademodul in den Arbeitsspeicher zu laden, kann man das Makro LOAD in den folgenden vier Formen verwenden:

```
LOAD   EP=symbol
LOAD   EPLOC=adr
LOAD   EPLOC=(Ø)
LOAD   EPLOC=(r)
```

Im Operanden EP ist ein Eingangspunkt des zu ladenden Lademoduls anzugeben. Wenn man den Operanden EPLOC benutzen will, so muß unter adr die Adresse eines Doppelwortes angegeben werden, in das der symbolische Name des Eingangspunktes hineingestellt wurde.

Dieses Makro bewirkt, daß der gewünschte Modul in den Hauptspeicher geladen und die Adresse seines Eingangspunktes in das Register 0 gestellt wird. Wenn nun die Programmablaufsteuerung diesem Modul übergeben werden soll, so benutzt man im allgemeinen die Instruktion

```
LR      15,Ø
```

und läßt auf diese das Makro CALL folgen.

Nachdem einmal ein Modul durch das LOAD-Makro in den Hauptspeicher gebracht worden ist, dann bleibt es dort bis zum Ende des Jobsteps oder solange, bis das Makro

```
DELETE EP=symbol
DELETE EPLOC=adr
DELETE EPLOC=(Ø)
DELETE EPLOC=(r)
```

gegeben wird. Das DELETE-Makro teilt dem Supervisor mit, daß das aufgeführte Modul nicht mehr länger im Hauptspeicher benötigt wird. Dieses Makro gibt im Register 15 einen Wert zurück, der das Ergebnis der Ausführung dieses Makos anzeigt:

0000 0000	Die Entfernung dieses Moduls war erfolgreich
0000 0004	Der angegebene Modul wurde nicht im Hauptspeicher gefunden

Ist mehr als ein LOAD-Makro für den gleichen Modul gegeben worden, so muß die gleiche Anzahl DELETE-Statements hinausgeschickt werden.

Um sowohl einen Modul in den Hauptspeicher einzulesen als auch ihm sofort die Programmablaufsteuerung zu übertragen, wird das Makro LINK in einer der beiden Formen

```
LINK   EP=symbol[,(arg1,···,argn)[,VL=1]][,ID=aa]
LINK   EPLOC=adresse[,(arg1,···,argn)[,VL=1]][,ID=aa]
```

eingesetzt. Der erste Operand entspricht dem gleichen Operanden des LOAD-Markos; für adresse kann auch (0) oder (r) geschrieben werden. Der zweite und dritte Operand entsprechen denen des CALL-Makros; für die Argumente ist auch die Registernotierung möglich.

Wenn das LINK-Makro verwendet wird, so wird der Inhalt des Registers 0 zerstört. Nach dem Eintritt in den aufgerufenen Modul enthält Register 15 die Adresse des Eingangspunktes und Register 1 zeigt auf die Argumentenliste, falls eine solche existiert. Die Register 2 bis 13 bleiben ungeändert, während Register 14 eine Rückkehradresse enthält, die auf einen Platz innerhalb des Supervisors verweist. Der verbundene Modul muß mit einer Verzweigung zu dem Platz enden, dessen Adresse im Register 14 festgehalten ist. Nach der Rückkehr zerstört das Steuerprogramm den Modul und übergibt die Programmablaufsteuerung an die auf die LINK-Makroinstruktion folgende Instruktion der rufenden Programmeinheit. Das DELETE-Makro ist nicht auf Moduln zu beziehen, die via des LINK-Makros geladen wurden. Das LINK-Makro ist folglich im Endeffekt als Kombination eines LOAD-, eines CALL- und eines DELETE-Makros anzusehen.

Beim Suchen nach den Moduln wird zuerst STEPLIB durchforstet; gibt es keine solche Bibliothek, so wird JOBLIB herangezogen. Wird der Modul dort nicht gefunden, dann kommt SYS 1.LINKLIB an die Reihe.

Aufgabengruppe 62

1) Löse die Aufgabe 2 der Aufgabengruppe 59 für das Betriebssystem OS!

2) Löse die Aufgabe 3 der Aufgabengruppe 59 für das Betriebssystem OS!

3) Erweitere die Aufgabe 2) wie folgt:
   Ordne dem Zeitgeber ein bestimmtes Zeitintervall, z.B. 1 min. zu und zähle im Programm, wieviel Mal die Schleife während des vorgegebenen Zeitintervalls durchlaufen wurde.

4) Löse die Aufgabe 6 der Aufgabengruppe 59 unter Verwendung der FORTRAN-Sprache; für die Eingabe/Ausgabe ist die Verwendung der AL im OS vorzusehen.

5) Schreibe ein Programm, das eine Lochkarte einliest, auf der je eine ganze Zahl in den Spalten 1 bis 10 bzw. 11 bis 20 abgelocht ist. Wandle beide in Zahlen der Gleitpunktdarstellung mit erweiterter Genauigkeit um. Danach berechne den Ausdruck

$$z = \frac{x}{y} + xy$$

   Unter x ist hierbei die auf den Spalten 1 bis 10 abgelochte Zahl zu verstehen, y ist gleich der auf den Spalten 11 bis 20 abgelochten Zahl.

   Es ist eine Lochkarte mit folgendem Inhalt zu stanzen:
   Spalten 1 bis 10: x
   Spalten 11 bis 20: y
   Spalten 21 bis 40: z (mit einem Dezimalpunkt auf der Spalte 31).

   Nach Voraussetzung sind x und y positiv. Der Simulator für die erweiterte Genauigkeit wird nicht benötigt. Weiterhin ist anzunehmen, daß

$$x < y < 10000$$

   ist.

6) Die Aufgabe 5) ist wie folgt zu modifizieren: Die Voraussetzung, daß $y < 10\,000$ ist, entfällt. Wenn der Wert von z zu groß ist, um bei der Ausgabe in den vorgesehenen Platz hinein zu passen, ohne daß nach links ein Abschneiden erfolgt, so sollen alle für z vorgesehenen Spalten mit der Ziffer 9 gefüllt werden.

23. VSAM

23.1. Die Steuerblöcke von VSAM

VSAM benutzt drei Steuerblöcke. Jede Datei benötigt zunächst einmal einen Zugriffssteuerblock[1)], der durch das Makro ACB installiert wird. Dieses Makro sieht wie folgt aus:

```
name      ACB    operand[,operand]...
```

Die Operanden sind in der Übersicht von Abb. 23.1. aufgeführt.

Eintragung	*Bedeutung*	*Verwendung*
ACBLEN=n	Länge des Zugriffssteuerblockes (ACB)	S T *
ATRB=attribute	--	T
AVSPAC=n	Verfügbarer Speicherplatz für die Datenkomponente	S T
BUFND=n	Anzahl der Datenpuffer; Standardwert für n gleich 2; Minimalanzahl gleich der um 1 erhöhten Abfragezahl	A M S T *
BUFNI=n	Anzahl der Indexpuffer; Standardwert gleich 1	A M S T *
BUFNO=n	Anzahl der Puffer für die gerade benutzte Komponente	S T
BUFSP=n	Von allen kombinierten Puffern belegter Speicherplatz	A M S T *
CATALOG={Yes / NO}	ACB bezieht sich auf eine katalogisierte Datenmenge	A M T ¤
CINV=n	Umfang des Kontrollintervalles	S T
DDNAMF=ddname	Name des DLBL- bzw. DD-Statements (8 Bytes lang); als Standardname gilt der des ACB-Makros	A M S T *[2]
ENDRBA=n	Relative Byte Adresse (RBA) des letzten benutzten Bytes in der Datenkomponente	S T ¤
ERROR=n	Rückkehrcode vom letzten OPEN- oder CLOSE-Statement	S T *
EXLST=name	Adresse des Makros EXLST; gleich Null, wenn diese fehlt	A M S T *
FS=n	Prozentuale freie Kontrollintervalle pro Kontrollbereich in der Datenkomponente (0 bei der Indexkomponente)	S T
KEYLEN=n	Länge des Schlüsselfeldes in Datensätzen	S T
LRECL=n	Maximale Satzlänge	S T

Abb. 23.1. Der Aufbau des Zugriffssteuerblockes ACB (1. Teil)

1) Im angloamerikanischen Sprachraum als „Access Control Block", abgekürzt ACB bezeichnet.

Eintragung	*Bedeutung*	*Verwendung*
MACRF=operanden	siehe Abb. 23.2.	A M T
NCIS=n	Anzahl der gesplitteten Kontrollintervalle (0 bei der Indexkomponente)	S T
NDELR=n	Anzahl der entfernten Sätze (0 bei der Indexkomponente)	S T
NEXCP=n	Anzahl der EXCP-Makros, die nach dem Eröffnen hinausgeschickt worden sind	S T
NEXT=n	Anzahl der zugeordneten Speicherbereiche	S T
NINSR=n	Anzahl der zur Datenkomponente hinzugefügten Sätze (0 bei der Indexkomponente)	S T
NIXL=n	Anzahl der Indexstufen	S T
NLOGR=n	Anzahl der Datensätze in der Datenkomponente (0 bei der Indexkomponente)	S T
NRETR=n	Anzahl der Sätze der Datenkomponente, zu denen ein Zugriff erfolgt (0 bei der Indexkomponente)	S T
NSSS=n	Anzahl der gesplitteten Kontrollbereiche in der Datenkomponente (0 bei der Indexkomponente)	S T
NUPDR=n	Gesamtzahl der aufdatierten Sätze in der Datenkomponente (0 bei der Indexkomponente)	S T
OFLAGS=OPEN	Prüfung, ob Datenmenge eröffnet ist	T
PASSWD=name	Adresse des Feldes, das den Datenschutzcode (Paßwort) enthält	A M S T *
RKP=n	Distanzadresse des Schlüsselfeldes vom Satzbeginn an	S T
STMST=name	Systemzeitangabe	S T 2
STRNO=n	Maximale Anzahl von Abfragen, für die VSAM sich die Position des Datensatzes in der Datenmenge merken soll	A M S T *

Es bedeuten:	A	Verwendung als Operand beim Makro ACB
	M	Verwendung beim Makro MODCB, auf ACB bezogen
	S	Verwendung beim Makro SHOWCB, auf ACB bezogen
	T	Verwendung beim Makro TESTCB, auf ACB bezogen
	*	Verwendung bei SHOWCB oder TESTCB nur bei geöffneten Dateien
	¤	Anwendung nur im Betriebssystem OS
	2	Acht Bytes langes Feld

Abb. 23.1. Der Aufbau des Zugriffssteuerblockes ACB (2. Teil)

Nur die mit einem A versehenen Operanden können als Teil des ACB-Makros codiert werden. Eintragungen, die mit einem S oder T gekennzeichnet sind, werden zur Eröffnungszeit der betreffenden Datenmenge hinzugestellt, wobei diese Informationen entweder aus dem VSAM-Katalog oder aus den Kennsätzen stammen. In den meisten Fällen reichen die Standardvorgaben aus. Das gilt nicht für den Schlüsselwortoperanden EXLST, der stets dann codiert werden muß, wenn eine Ausgangsliste benötigt wird. Der Operand MACRF wird sehr oft codiert, weil die Standardvorgaben für ihn vielfach nicht ausreichen. Abb. 23. 2 zeigt die für MACRF vorhandenen nachgeordneten Operanden (Unteroperanden). Unteroperanden werden voneinander durch Kommas getrennt; der gesamte Operand wird in Klammern eingeschlossen. Bei Fehlen des MACRF-Operanden gilt als Standardvorgabe

```
MACRF=(KEY,SEQ,IN)
```

Unteroperand	*Bedeutung*	*Verwendung*		
		A	M	T
KEY	Ermöglicht die Verwendung von OPTCD=KEY im Makro RPL	A	M	T
ADR	Ermöglicht die Verwendung von OPTCD=ADR im Makro RPL	A	M	T
SEQ	Ermöglicht die Verwendung von OPTCD=SEQ im Makro RPL			
SKP	Ermöglicht die Verwendung von OPTCD=SKP im Makro RPL			
DIR	Ermöglicht die Verwendung von OPTCD=DIR im Makro RPL			
IN	Ermöglicht die Verwendung der folgenden Makros: OPEN, CLOSE, GET, POINT, ENDREQ und a) zusätzlich im Betriebssystem DOS: TCLOSE b) zusätzlich im Betriebssystem OS: CHECK	A	M	T
OUT	Ermöglicht die Verwendung der folgenden Makros: OPEN, CLOSE, GET, POINT, ENDREQ, PUT, ERASE und a) zusätzlich im Betriebssystem DOS: TCLOSE b) zusätzlich im Betriebssystem OS: CHECK	A	M	T

Anmerkungen:
1) Die Standards der einzelnen Gruppen sind unterstrichen
2) Es bedeuten

A	Verwendung beim Makro ACB
M	Verwendung beim Makro MODCB, auf ACB bezogen
T	Verwendung beim Makro TESTCB, auf ACB bezogen

Abb. 23.2. Die Unteroperanden des Operanden MACRF vom Makro ACB

Ein typisches ACB-Makrostatement besitzt z. B. folgendes Aussehen:

```
T        ACB    MACRF=(ADR,SEQ,OUT),EXLST=XT
```

Eine Ausgangsliste[1], mit der verschiedene Bedingungen gehandhabt werden können, kann im Bedarfsfall definiert werden. Hierzu ist das Makro EXLST geschaffen worden. Allgemein sieht es wie folgt aus:

```
name    EXLST operand[,operand]···
```

Die für dieses Makro benutzbaren Operanden sind in Abb. 23.3. zusammengestellt.

Feldeintragung	*Bedeutung*	*Verwendung*
EODAD=wert	Adresse der Dateienderoutine	E M S T
EXCPAD=wert	Adresse der Ausgangsroutine für EXCP	E M S T 1
EXLLEN=n	Länge des Steuerblockes von EXLST	S T
JRNAD=wert	Adresse der Ausgangsroutine für die Journalführung (Protokollierung)	E M S T 2
LERAD=wert	Adresse der Ausgangsroutine zur Behandlung logischer Fehler	E M S T
SYNAD=wert	Adresse der Ausgangsroutine zur Behandlung physikalischer Fehler	E M S T

Anmerkungen:

1) Für wert kann dabei eingesetzt werden

```
(adresse,A)
(adresse,A,L)
(adresse,L,A)
(adresse,N)
(adresse,N,L)
(adresse,L,N)
Ø
```

Der Wert Ø ist nur für Testzwecke (TESTCB) möglich

2) Es bedeuten:

E	Verwendung beim Makro EXLST
M	Verwendung beim Makro MODCB, auf EXLST bezogen
S	Verwendung beim Makro SHOWCB, auf EXLST bezogen
T	Verwendung beim Makro TESTCB, auf EXLST bezogen
1	Nur im Betriebssystem DOS gültig
2	Nur im Betriebssystem OS gültig

Abb. 23.3. Der Aufbau des Steuerblockes für die Ausgangsliste EXLST

Nur diejenigen Eintragungen, die durch ein E gekennzeichnet sind, können als Teil des EXLST-Makros codiert werden. Als Unteroperanden von wert können L, A und N benutzt werden. Ihre Bedeutung geht aus den nachfolgenden Ausführungen hervor. Erscheint L nicht als Unteroperand, so ist adresse die Adresse des Eingangspunktes der betreffenden Ausgangsroutine. Wird dagegen L als Unteroperand genannt, so ist adresse die Adresse eines acht Bytes langen Feldes, das den Namen einer Phase aus

1) Im angloamerikanischen Sprachraum als „Exit List" bezeichnet

der Bibliothek ausführbarer Programme (Betriebssystem DOS) oder das den Namen eines Gliedes aus einer der Bibliotheken STEPLIB, JOBLIB oder SYS1. LINKLIB (Betriebssystem OS) enthält. Die genannte Phase oder das genannte Glied wird dann geladen und ausgeführt, wenn die den Ausgang kennzeichnende Bedingung auftritt. Wird A codiert, so wird der Ausgang betreten, wenn die Bedingung zustande gekommen ist; bei Niederschrift von N ist der Ausgang nicht aktiv, er kann zufolge selbst bei Auftreten der Bedingung nicht eingeschlagen werden. Während des Programmablaufes können die Fälle A und N gegenseitig ausgetauscht werden. Zusätzliche Ausgänge können jedoch nicht mehr in die vom EXLST-Makro geschaffene Ausgangsliste aufgenommen werden.

Ein einziges EXLST-Makro kann mehr als einem ACB-Makro zugeordnet werden.

Wenn der Ausgang EODAD nicht aktiv ist, so wird beim Auftreten der Dateiendebedingung die Programmablaufsteuerung an die Routine der LERAD-Ausgangs übergeben, falls eine solche aktiv ist.

In der Abb. 23.4. sind die Registerinhalte zusammengefaßt, so wie sie sich beim Eintritt in eine Ausgangsroutine zeigen.

Register	Inhalt
0	Unbestimmt
1	Adresse der Anforderungsparameterliste RPL. Bei JRNAD- und EXCPAD-Ausgangsroutinen: Adresse einer Parameterliste, deren erste Eintragung die Adresse der Anforderungsparameterliste ist. Bei Rückkehr zum VSAM muß dieses Register inhaltlich unverändert sein.
2 bis 12	Die gleichen wie beim RPL-Makro; unbestimmt bei JRNAD.
13	Adresse des Rettungsgebietes im Benutzerprogramm. Dieses Gebiet wird von VSAM benutzt und kann daher nicht von der Ausgangsroutine benutzt werden. Bei Rückkehr nach VSAM muß dieses Register unverändert bleiben.
14	Rückkehradresse nach VSAM
15	Adresse der Ausgangsroutine

Abb. 23.4. Registerinhalte beim Eintritt in eine Ausgangsroutine

Eine Ausgangsroutine braucht nicht unbedingt zu VSAM zurückzukehren. Ist das jedoch vorgesehen, so müssen die Inhalte der Register 1 und 13 aufbewahrt werden. Bei Fehlen einer Ausgangsroutine wird ins Register 15 ein Rückkehrcode gestellt, und es wird anschließend die Nachfolgeinstruktion ausgeführt. Nach Rückkehr von einer Ausgangsroutine kommt ebenfalls die nächste Instruktion zum Zuge; ein Rückkehrcode steht jedoch nicht zur Verfügung.

Ein typisches EXLST-Makrostatement besitzt folgendes Aussehen:

```
XT          EXLST EODAD=(DATEND99,A)
```

Mit Hilfe des Makros RPL wird der dritte der Steuerblöcke, die Anforderungsparameterliste[1], generiert. Das Makro sieht allgemein wie folgt aus:

```
name      RPL    operand [,operand] ...
```

Die benutzbaren Operanden sind in Abb. 23.5. zusammengefaßt.

Eintragung	*Bedeutung*	*Verwendung*
ACB=name	Adresse des Zugriffssteuerblockes ACB	R M S T
AREA=name	Adresse des Arbeitsgebietes	R M S T
AREALEN=n	Länge des Arbeitsgebietes	R M S T
ARG=name	Adresse des Suchargumentes	R M S T
ECB=name	Adresse des Ereignissteuerblockes ECB	R M S T 2
FDBK=n	Rückmeldewert bei bestimmten Fehlertypen einer EA-Operation	S T
IO=COMPLETE	Test auf Vollendung asynchroner Operationen	T 2
KEYLEN=n	Länge des generischen Schlüssels	R M S T
MSGAREA=name	Adresse des Speichergebietes, in das VSAM physikalische Fehlernachrichten hineinstellt	R M S T 2
MSGLEN=n	Länge des durch MSGAREA festgelegten Gebietes; erforderlich bei Vorhandensein des Operanden MSGAREA	R M S T 2
NXTRPL=name	Adresse der nächsten Anforderungsparameterliste, die an diese angekettet ist	R M S T
OPTCD=optionen	siehe Abb. 23.6.	R M T
RBA=n	Relative Byteadresse des laufenden Satzes	S T
RECLEN=n	Länge des Datensatzes	R M S T
RPLLEN=n	Länge des Steuerblockes für die Anforderungsparameterliste RPL	S T

Es bedeuten:

R	Verwendung beim Makro RPL
M	Verwendung beim Makro MODCB, auf RPL bezogen
S	Verwendung beim Makro SHOWCB, auf RPL bezogen
T	Verwendung beim Makro TESTCB, auf RPL bezogen
2	nur gültig im Betriebssystem OS

Abb. 23.5. Der Aufbau des Steuerblockes für die Anforderungsparameterliste RPL

Nur die mit dem Buchstaben R gekennzeichneten Angaben können als Teil des RPL-Makros codiert werden. Die Verarbeitung, die exakt ablaufen soll, wird durch Unteroperanden vom Schlüsselwortoperanden OPTCD bestimmt. Die Unteroperanden können wir der Abb. 23.6. entnehmen.

[1] Im angloamerikanischen Sprachraum als „Request Parameter List", abgekürzt RPL bezeichnet

Gruppe	*Unteroperand*	*Bedeutung*	*Verwendung*
1	KEY	Zugriff über den Schlüssel; nur für Dateien in Schlüsselfolge	R M T
	ADR	Zugriff über die Adresse	R M T
2	SEQ	Sequentieller Zugriff	R M T
	SKP	Überspringen des sequentiellen Zugriffs; nur für Dateien in Schlüsselfolge	R M T
	DIR	Direkter Zugriff	R M T
3	NUP	Kein Aufdatieren; PUT bezieht sich auf einen neuen Satz; bei DIR wird die Stellung nicht gemerkt	R M T
	NSP	Kein Aufdatieren; PUT bezieht sich auf einen neuen Satz; die Stellung wird gemerkt (nur bei Angabe von DIR)	R M T
	UPD	Aufdatieren erlaubt; PUT bezieht sich auf den aufzudatierenden Satz, die Stellung wird gemerkt	R M T
4	MVE	Durch GET wird ein Satz in das Arbeitsgebiet übertragen; durch PUT ein Satz aus dem Arbeitsgebiet	R M T
	LOC	Das Makro GET beläßt den Satz im EA-Puffer; für PUT nicht gültig	R M T
5	KEQ	Der Satzschlüssel muß gleich dem Suchargument sein; nicht bei SEQ angebbar; Angabe muß sowohl für GET als auch für POINT erfolgen	R M T
	KGE	Satzschlüssel muß größer als oder gleich dem Suchargument sein, sonst wie bei KEQ	R M T
6	FKS	Der Satzschlüssel wird in gesamter Länge mit dem Suchargument verglichen	R M T
	GEN	Generischer Schlüssel; der Vergleich erfolgt in der Länge, die bei KEYLEN angegeben ist	R M T
7	SYN	Synchroner Zugriff; die Operation wird vollendet, bevor die nächste Instruktion ausgeführt wird	R M T 2
	ASY	Asynchroner Zugriff; die Operation ist noch nicht beendet, wenn die nächste Instruktion ausgeführt wird. Das Makro CHECK muß an der Stelle gegeben werden, an der auf die Vollendung der Operation gewartet werden muß.	R M T 2

Es bedeuten:

R	Verwendung beim Makro RPL
M	Verwendung beim Makro MODCB, auf RPL bezogen
T	Verwendung beim Makro TESTCB, auf RPL bezogen
2	nur gültig im Betriebssystem OS

Abb. 23.6. Unteroperanden des Operanden OPTCD vom Makro RPL

Zu einer bestimmten Zeit kann stets nur ein Unteroperand jeder Gruppe aktiv sein. Fehlt der Schlüsselwortoperand OPTCD, so gilt als Standard

```
OPTCD=(KEY,SEQ,NUP,MVF,KEQ,FKS,SYN)
```

Nicht alle Operanden sind für jede Operation erforderlich. Z. B. braucht KEYLEN für den sequentiellen Zugriff nicht angegeben zu werden, wenn es sich um eine Datei in Zugangsfolge handelt. Nicht benötigte Operanden werden einfach ignoriert. Ein typisches RPL-Statement sieht wie folgt aus:

```
RT        RPL   ACT=T,AREALEN=19,RECLEN=19
                AREA=WKLENT,OPTCD=(ADR,SEQ,NUP,MVE)
```

Dieses Makro kann nur in Verbindung mit den o. a. Musterstatements für ACB und EXLST gebraucht werden.

23.2. Das Eröffnen und Schließen von VSAM-Dateien

VSAM-organisierte Dateien werden genau so eröffnet wie andere Dateien. Sie können zusammen mit anderen Dateien in einem Statement eröffnet werden. Ein Rückkehrcode ist im Register 15 zu finden. Dieser Code ist gleich Null, wenn alle genannten Dateien erfolgreich eröffnet wurden. Er ist jedoch ungleich Null, wenn mindestens eine Datei nicht eröffnet werden konnte oder wenn beim Eröffnen ein Fehler aufgetreten ist.[1)]

Das CLOSE-Statement wird genau so verwendet wie bei anderen Dateien auch. Der im Register 15 zurückgegebene Rückkehrcode sagt bei Null aus, daß das Abschließen funktioniert hat, ein Code ungleich Null verweist darauf, daß nicht alle Dateien ordnungsgemäß abgeschlossen werden konnten.[2)]

In den beiden Betriebssystemen OS und DOS ist je eine zusätzliche Form des CLOSE-Statements benutzbar. Sie lautet:

```
OS  |   CLOSE (dateiname,...),TYPE=T

DOS |   TCLOSE dateiname,...
```

Unter dateiname müssen bei diesen Statements stets die Namen von VSAM-Dateien erscheinen. Die durch diese Statements kreierte Variante läuft zwar prinzipiell auf das reguläre Abschließen hinaus, doch bleibt die Datei für nachfolgende Verarbeitungsoperationen geöffnet. Das sogenannte zeitweilige Abschließen dient zum Schreiben von Schlußsätzen, zum Aufdatieren des VSAM-Kataloges usw. Im allgemeinen wird dadurch die Datei und der VSAM-Katalog gegenüber Programmfehlern geschützt, die im restlichen Verlauf des Programmes auftauchen könnten. Bei langlaufenden Programmen kann darin eine höchst nutzvolle Vorsichtsmaßnahme gesehen werden.

Wenn eine Datei eröffnet wird, so wird sie gleichzeitig auf den ersten mit Schlüssel und mit der relativen Byteadresse von Null versehenen Satz eingestellt. Alle nachfolgenden Einstellungen hängen von den durchgeführten Operationen ab. Jede Anforderungsparameterliste verwaltet ihre eigene Einstellposition bezüglich der Sätze in der Datei.

1) Im Betriebssystem OS werden alle im OPEN-Statement angegebenen Optionen bei VSAM-Dateien ignoriert
2) Im Betriebssystem OS werden alle im CLOSE-Statement angegebenen Optionen bei VSAM-Dateien ignoriert

23.3. Das Modifizieren von Steuerblöcken

Durch das Makro

```
MODCB  block=adresse, {schlüsselwort=wert}···
```

kann der Inhalt von Steuerblöcken geändert werden. Unter block kann ACB, EXLST oder RPL erscheinen, unter adresse ist der dem Block gegebene Name zu verstehen. Es können soviele Schlüsselwortoperanden angegeben werden, wie man für erforderlich erachtet. Durch Spezifizierung aller Schlüsselwortoperanden in diesem Makro kann man sich der Notwendigkeit entziehen, irgendwelche Operanden in den ACB- und RPL-Makros angeben zu müssen. Beim MODCB-Makros gibt es keine Standards.

Beim Steuerblock ACB können neue Angaben zum MACRF-Operanden hinzugefügt werden, bestehende jedoch nicht beseitigt werden. Für den EXLST-Steuerblock kann zusätzlich noch

```
routine=A
```

oder

```
routine=N
```

genannt werden. Damit kann der Aktivitätsstatus der betreffenden Ausgangsroutine geändert werden. Wird die Adresse einer Ausgangsroutine geändert, so gilt weder A noch N als Standardwert. Wird für den RPL-Steuerblock ein Unteroperand für OPTCD festgelegt, so wird eine bestehende Option der gleichen Gruppe getilgt.

Am häufigsten wird das Makro MODCB dazu benutzt, um die Satzlänge in einer Anforderungsparameterliste zu ändern, wenn Sätze verschiedener Längen geschrieben werden müssen.

Ein Vorteil des MODCB-Makros gegenüber den Definitionsmakros ACB, EXLST und RPL besteht darin, daß für alle Operanden eine Registernotierung (r) möglich ist. Für r können dabei die Werte von 2 bis einschließlich 12 vorliegen. Die Registernotierung kann freilich nur für Operanden benutzt werden, für die numerische Werte oder Adressen angegeben werden können. Für die letzteren können zusätzlich auch Adreßausdrücke vom A-Typ spezifiziert werden.

Ein typischen MODCB-Statement lautet:

```
MODCB  RPL=(3),RECLEN=(4)
```

Hierbei muß das Register 3 die Adresse der Anforderungsparameterliste enthalten, bei der die ihn ihr enthaltene Satzlänge geändert werden soll. Die neue Satzlänge ist im Register 4 zu finden.

23.4. Auskunft über Steuerblöcke

Durch das Makro

```
SHOWCB  block=adresse,AREA=name,LENGTH=n,
        [OBJEKT={DATA|INDEX}],FIELDS=(schlüsselwort,···)
```

wird im Gebiet name eine Tabelle der Länge n zusammengestellt, in der die augenblicklichen Werte der unter FIELDS genannten Schlüsselwortoperanden verzeichnet sind, die zu dem Block gehören, dessen Name unter adresse genannt ist. Unter block kann entweder ACB oder EXLST oder RPL stehen. Jedes Feld, dessen Inhalt angezeigt werden

soll, erfordert die Bereitstellung eines vollen Wortes, bei einigen sogar von zwei Wörtern (siehe hierzu Abb. 23.1.). Ein typisches Makro dieser Art sieht wie folgt aus:

```
SHOWCB   RPL=(3),AREA=GEBIET,LENGTH=8,
         FIELDS=(RECLEN,FDBK)
```

Dieses Statement bewirkt folgendes: Die augenblickliche Satzlänge wird nach GEBIET gebracht und der augenblickliche Rückkehrcode nach GEBIET+4. Die Adresse der Anforderungsparameterliste, für die die Auskunft eingeholt werden soll, ist im Register 3 zu finden. Wie beim Makrostatement MODCB ist eine Registernotierung für adresse, name und n möglich; Adreßausdrücke können für adresse und name gebraucht werden.

Am häufigsten wird SHOWCB dazu benutzt, um zu der Länge des Satzes zu gelangen, der gerade geholt wurde und um den Rückkehrcode zu prüfen und zu untersuchen, wenn eine Operation anormal beendet wurde.

23.5. Das Überprüfen von Werten in Steuerblöcken

Das Makro

```
TESTCB   block=adresse [,OBJECT= {DATA|INDEX}] [,ERET=name],
         schlüsselwort=wert
```

geht in seiner Wirkungsweise über die des Makros SHOWCB hinaus. Im optionalen Operanden ERET ist die Adresse einer Routine anzugeben, an die die Programmablaufsteuerung übertragen werden soll, wenn das Makro nicht erfolgreich ausgeführt werden kann und nur ein Schlüsselwort aufgeführt ist. Folgender Vergleich wird durchgeführt: Wert im TESTCB-Makro gegen den entsprechenden Wert im Steuerblock. Dabei wird der Bedingungsschlüssel in Abhängigkeit vom Vergleichsergebnis gesetzt, und zwar ob der bei schlüsselwort angegebene Wert kleiner als, gleich oder größer als der im Makro TESTCB spezifizierte Wert ist. Für Elemente, bei denen unter wert nur ein Code steht, wie OFLAGS=OPEN, läuft der Vergleich nur auf gleich und ungleich hinaus. Nach dem Vergleich erfolgt der Übergang zur Nachfolgeinstruktion, die in der Regel zum Testen des Bedingungsschlüssels dienen wird.

Typische Makros sehen wie folgt aus:

```
TESTCB RPL=(3),RECLEN=78
TESTCB RPL=RT,RBA=(8)
```

Beim ersten Beispiel wird die aktuelle Satzlänge mit 78 verglichen.

Auf den ACB-Steuerblock kann sich das Schlüsselwort ATRB beziehen, wobei es wie folgt zu codieren ist:

```
ATRB=(attribut,···)
```

Erlaubt ist die Angabe der folgenden Attribute:

ESDS	Datenmenge in Zugangsfolge
KSDS	Datenmenge in Schlüsselfolge
REPL	Wiederholung von Indexteilen
SSWD	Index stößt an die Datenmenge an
WCK	Vorgesehene Schreibprüfung

Durch TESTCB wird der Bedingungsschlüssel auf 0 (gleich) gesetzt, wenn alle Attribute, die in TESTCB genannt sind, für die zu betrachtende Datenmenge korrekt sind.

Für die beiden Makros SHOWCB und TESTCB sind auch noch andere Formen möglich:

```
SHOWCB AREA=name,LENGTH=n,FIELDS=(···)
TESTCB [ERET=name,] schlüsselwort=n
```

In dieser Form sind die einzigen erlaubten Schlüsselwörter die folgenden: ACBLEN, EXLLEN und RPLLEN. Alle Steuerblöcke einer bestimmten Form besitzen die gleiche Länge.

Bei Ausführung der sich hinter den Makros MODCB, SHOWCB und TESTCB Operationen wird ein Rückkehrcode ins Register 15 gestellt. Bei erfolgreicher Ausführung beträgt dieser Null, ansonsten kommt es zum Auftreten eines von Null verschiedenen Rückkehrcodes.

23.6. Makrostatements für die Verarbeitung

VSAM-Dateien werden verarbeitet, wenn die folgenden Makros niedergeschrieben werden:

```
GET     RPL=name
PUT     RPL=name
POINT   RPL=name
ERASE   RPL=name
CHECK   RPL=name
ENDREQ  RPL=name
```

Das Makro CHECK ist nur im Betriebssystem OS verfügbar. Die Angabe von name kann auch in Registernotierung erfolgen, ein Adreßausdruck ist ebenfalls gestattet.

Durch GET wird zu einem Satz in der Datei zugegriffen, durch PUT wird entweder ein neuer Satz geschrieben oder ein zuvor gelesener Satz wieder in die Datei zurückgestellt. Durch POINT wird nichts mit der Datei getan, dieses Makro dient einfach nur zur Einstellung der Anforderungsparameterliste auf eine bestimmte Position der Datei, entweder durch die relative Byteadresse oder durch den Schlüssel. Diese Position kann als Startpunkt für nachfolgende sequentielle Operationen herangezogen werden. Bei Verwendung des Makros ERASE wird ein zuvor gelesener, mit einem Schlüssel versehener Satz aus der Datei entfernt. Durch das ENDREQ-Makro wird die Positionsinformation gestrichen, die in der Anforderungsparameterliste zurückgehalten wurde; es kann ebenfalls eine EA-Operation abbrechen, die noch nicht vollständig abgelaufen ist. Das CHECK-Makro bewirkt, daß die Verarbeitung bis zur Vervollständigung der gerade ablaufenden EA-Operation warten muß.

Die spezielle Art der Verarbeitung hängt von den im Operanden OPTCD aufgeführten Optionen ab, die ja bekanntlich in die Anforderungsparameterliste eingehen. Die erlaubten Kombinationen sind in der Abb. 23. 7. zusammengestellt.

Bei der sequentiellen Verarbeitung wird Satz auf Satz geholt, entweder in der Schlüsselfolge (OPTCD=KEY) oder in der Folge der relativen Byteadressen (OPTCD=ADR). Zu einer Datenmenge in Schlüsselfolge kann in jeder Form zugegriffen werden. Das POINT-Makro kann jederzeit dazu benutzt werden, um die Anforderungsparameterliste neu zu positionieren; das nächste GET-Statement greift dann auf den Satz zu, auf den gezeigt wird. Bei der direkten Verarbeitung muß der Benutzer entweder den Schlüssel (OPTCD=KEY) oder die relative Byteadresse (OPTCD=ADR) desjenigen Satzes

Dateien in							
Schlüsselfolge				Zugangsfolge			
Optionen	GET	PUT	ERASE	Optionen	GET	PUT	ERASE
KEY SEQ NUP MVE	G1 N2	P5 N2	E0	ADR SEQ NUP MVE	G2 N1	P4 N1	E0
KEY SEQ NUP LOC	G1 N2	P0	E0	ADR SEQ NUP LOC	G2 N1	P0	E0
KEY SEQ UPD MVE	G1 N4	P2 N2	E1 N2	ADR SEQ UPD MVE	G2 N3	P3 N1	E1 N1
KEY SEQ UPD LOC	G1 N4	P0	E1 N2	ADR SEQ UPD LOC	G2 N3	P0	E1 N1
KEY SKP NUP MVE	G5 N2	P5 N2	E0	ADR DIR NUP MVE	G4 N0	P4 N0	E0
KEY SKP NUP LOC	G5 N2	P0	E0	ADR DIR NUP LOC	G4 N1	P0	E0
KEY SKP UPD MVE	G5 N4	P2 N2	E1 N2	ADR DIR NSP MVE	G4 N1	P4 N1	E0
KEY SKP UPD LOC	G5 N4	P0	E1 N2	ADR DIR NSP LOC	G4 N1	P0	E0
KEY DIR NUP MVE	G3 N0	P1 N0	E0	ADR DIR UPD MVE	G4 N3	P3 N0	E1 N0
KEY DIR NUP LOC	G3 N2	P0	E0	ADR DIR UPD LOC	G4 N3	P0	E1 N0
KEY DIR NSP MVE	G3 N2	P1 N2	E0				
KEY DIR NSP LOC	G3 N2	P0	E0				
KEY DIR UPD MVE	G3 N4	P2 N0	E1 N0				
KEY DIR UPD LOC	G3 N4	P0	E1 N0				
ADR SEQ NUP MVE	G2 N1	P0	E0				
ADR SEQ NUP LOC	G2 N1	P0	E0				
ADR SEQ UPD MVE	G2 N3	P3 N1	E1 N1				
ADR SEQ UPD LOC	G2 N3	P0	E1 N1				
ADR DIR NUP MVE	G4 N0	P0	E0				
ADR DIR NUP LOC	G4 N1	P0	E0				
ADR DIR NSP MVE	G4 N1	P0	E0				
ADR DIR NSP LOC	G4 N1	P0	E0				
ADR DIR UPD MVE	G4 N3	P3 N0	E1 N0				
ADR DIR UPD LOC	G4 N3	P0	E1 No				

Es bedeuten:

G1	Zugriff zum nächsten Satz in Schlüsselfolge
G2	Zugriff zum nächsten Satz nach relativer Byteadresse
G3	Zugriff zum Satz, dessen Schlüssel in ARG enthalten ist
G4	Zugriff zum Satz, dessen relative Bytadresse in ARG enthalten ist
G5	Zugriff zum Satz, dessen Schlüssel in ARG enthalten ist, wobei vorzusehen ist, daß der in ARG enthaltene Schlüssel größer als oder gleich dem Schlüssel der vorhergehenden GET-Operation ist
P0	Nicht erlaubt
P1	Hinzufügen eines neuen Satzes in Schlüsselfolge
P2	Bei N4: Ersetzen des zuvor geholten Satzes durch einen neuen Satz beliebiger Länge
P3	Bei N3: Ersetzen des zuvor geholten Satzes durch einen neuen Satz gleicher Länge
P4	Hinzufügen eines neuen Satzes am Dateiende
P5	Bei N2: Hinzufügen eines neuen Satzes in Schlüsselfolge, wobei dessen Schlüssel größer als der Schlüssel des vorhergehenden Satzes sein muß
E0	Nicht erlaubt
E1	Bei N3 oder N4: Entfernen des zuvor geholten Satzes
N0	Kein UPDATE-Modus; keine Aufbewahrung einer Einstellposition
N1	Kein UPDATE-Modus; keine Aufbewahrung der relativen Byteadresse
N2	Kein UPDATE-Modus; Aufbewahrung des Schlüssels und der relativen Byteadresse
N3	UPDATE-Modus; Aufbewahrung der relativen Byteadresse
N4	UPDATE-Modus; Aufbewahrung des Schlüssels und der relativen Byteadresse

Abb. 23.7. Übersicht über die Verarbeitung mit den Makros GET, PUT und ERASE

nennen, zu dem zugegriffen oder der zur Datenmenge hinzugefügt werden soll. Bei der direkten Verarbeitung wird die Dateipositionierung in der Regel nicht festgehalten, es sei denn, daß OPTCD=NSP angegeben ist. Wird die Dateipositionierung (Einstellposition) aufbewahrt, so kann der Übergang zur sequentiellen Verarbeitung erfolgen, ohne daß das Makro POINT benutzt wird. Die sequentielle Verarbeitung startet dann bei dem Satz, der sich an denjenigen anschließt, zu dem direkt zugegriffen wurde. Die sequentielle Verarbeitung mit Überspringen (SKIP), d.h. ein sprungweises Vorgehen, ist wie die direkte aufgezogen; sie steht jedoch nur für den Zugriff über Schlüssel zur Verfügung, und die Schlüssel müssen außerdem in aufsteigender Reihenfolge vorliegen. Auf jeden Fall leistet sie mehr als die direkte Verarbeitung.

Im Aufdatierungsmodus (OPTCD=UPD) kann nach jedem Zugriff ein PUT-Statement hinausgeschickt werden, das dann den geholten Satz ersetzt. Beim Zugriff über Schlüssel kann ein geholter Satz durch ERASE entfernt werden. In allen anderen Verarbeitungsarten wird durch PUT ein neuer Satz hinzugefügt. Bei Dateien in Zugangsfolge kommt der neue Satz aus Dateiende, bei Dateien in Schlüsselfolge dorthin, wohin er nach dem Schlüssel gehört. VSAM bestimmt den Schlüssel aus seiner im Satz eingebetteten Lage.

Meist werden die Sätze in ein Arbeitsgebiet übertragen und von dort auch geschrieben (OPTCD=MVE). Beim Lesen und nur beim Lesen mit OPTCD=LOC kann der Benutzer den Satz auch im EA-Gebiet verarbeiten. Die Verarbeitung im Lokalisierungsmodus zur Vorbereitung auf die Ausgabe (PUT-Statements) ist nicht erlaubt. Im Lokalisierungsmodus ist das unter dem Schlüsselwort AREA in der Anforderungsparameterliste anzugebende Gebiet als vierstelliges Gebiet anzugeben, in das VSAM die Adresse des im EA-Gebiet zu verarbeitenden Satzes hineinstellt. Für den Lokalisierungsmodus gilt also für die Länge die Angabe AREALEN=4.

Durch die Makroausführung wird die Operation gestartet. Ist OPTCD=SYN angegeben, so wartet das Programm auf die Vollendung der Operation, ehe die nächstfolgende Instruktion in Angriff genommen wird. Es wird (jedoch nur im Betriebssystem DOS) unmittelbar nach Operationsbeginn der Ausgang EXCPAD eingeschlagen. Das Makro ENDREQ, in der Ausgangsroutine gebraucht, beendet abrupt die Operation, ohne daß sie vollständig abgelaufen ist; ein solches Vorgehen gilt als anormal. Im Betriebssystem OS kann OPTCD=ASY codiert werden. Bei dieser Vorgehensweise wird die Operation nur gestartet, die Programmablaufsteuerung geht aber dann sofort zur nächsten Instruktion über, d.h. es kommen während des Operationsablaufes andere Instruktionen zum Zuge. Ist es nun erforderlich, daß an einer bestimmten Programmstelle auf die Vollendung der Operation gewartet werden muß, so ist das CHECK-Makro hinauszuschicken. Der Operand ECB[1] bezieht sich bei diesem Makro auf einen Ereignissteuerblock. Dessen Bitstelle 0 wird vom System auf 1 gesetzt, wenn die Operation gestartet wird, nach Vollendung der Operation wird diese Bitstelle wieder auf 0 gesetzt. Der dieser Bitstelle zugewiesene Wert kann im Programm geprüft werden, bevor das CHECK-Makro benutzt wird.

Der Operand NXTRPL bei der Definition der Anforderungsparameterliste erlaubt die Verkettung mehrerer solcher Listen, die dann nacheinander von einem GET-, einem PUT-Statement abgearbeitet werden. Am Schluß aller Operationen wird die Einstellposition entsprechend der letzten ausgeführten Operation gesetzt. Allerdings ist die Angabe von NXTRPL nicht erlaubt, wenn OPTCD=LOC oder OPTCD=UPD spezifiziert ist. Alle Anforderungsparameterlisten in der Kette müssen im OPTCD-Operanden die gleichen Optionen besitzen. Sollte eine solche Kette anormal beendet werden, so werden

1) Abkürzung von „Event Control Block" (Ereignissteuerblock)

die nach der fehlerhaften noch verbleibenden Listen nicht mehr abgearbeitet. Auf eine solche Situation muß die Fehlerbehandlungsroutine des Benutzers eingehen.

Durch den Operanden mit dem Schlüsselwort STRNO wird die Maximalzahl von Einstellpositionen in Anforderungsparameterlisten genannt, die sich VSAM in jedem Augenblick merken soll. Wird mehr als eine Liste für die Datei benutzt, so kann es vorkommen, das die Zahl der Anforderungsparameterlisten mit Einstellinformationen größer ist als der im Operanden STRNO angegebene Wert. Das Makro ENDREQ kann dann dazu benutzt werden, um die Einstellinformationen von einer oder mehreren Listen zu vergessen; alternativ hierzu kann der Wert von STRNO auch vergrößert werden. Die Alternative mag nicht immer sinnvoll sein, da jede Vergrößerung des Wertes einen zusätzlichen Datenpuffer erfordert sowie 600 bis 1200 Bytes virtuelle Speicherplätze.

23.7. Fehlerbehandlung

Jedes der im Abschnitt 23.6. beschriebenen Makros liefert nach Ausführung der von ihm hervorgerufenen Operation einen Rückkehrcode im Register 15 ab. Es gibt folgende Rückkehrcodes:

Rückkehrcode	Bedeutung
00000000	Erfolgreiche Beendigung der Operation
00000004	Anforderungsparameterliste in Benutzung, jedoch Operation nicht gestartet
00000008	Logischer Fehler
0000000C	Physikalischer Fehler

Zusätzlich wird das bei FKBK genannte Feld in der RPL-Definition mit einem Rückmeldewert versorgt, aus dem dann beim Auftreten des Rückkehrcodes 8 oder C die Art des logischen oder des physikalischen Fehlers entnommen werden kann. Für logische Fehler sind die Werte in der Abb. 23.8. aufgeführt.

Beim Rückkehrcode C wird der unter SYNAD genannten Routine die Programmablaufsteuerung übertragen, falls eine solche im EXLST-Makro genannt ist. Im anderen Fall wird sofort die nächste Instruktion angezogen. Nach Rückkehr von der Ausgangsroutine wird die nächste Instruktion ausgeführt, der Rückkehrcode ist jedoch nicht mehr im Register 15 zu finden.

Dieselbe Situation entsteht beim Rückkehrcode 8 in bezug auf die unter LERAD genannte Ausgangsroutine. Eine Dateiendebedingung ohne Vorhandensein einer entsprechenden Ausgangsroutine (EODAD) führt ebenfalls zur Verursachung eines logischen Fehlers, wobei der Rückmeldewert 4 beträgt.

Fehlen Ausgangsroutinen für logische und physikalische Fehler, so wird der Rückmeldewert ins Register 0 gestellt. Bei Vorhandensein einer Ausgangsroutine geschieht das nicht.

Rückmeldewert	*Bedingung*
04	Dateiende beim sequentiellen Zugriff
08	Speicherungsversuch eines Satzes mit bereits vorhandenem Schlüssel
0C	Speicherungsversuch eines Satzes außerhalb der Reihenfolge
10	Kein Satz gefunden
14	Satz unter exklusiver Kontrolle einer anderen Anforderungsparameterliste
18	Satz ist auf nicht montiertem Datenträger gespeichert
1C	Kein Platz zur Erweiterung der Datei vorhanden
20	Die angegebene relative Byteadresse gehört zu keinem Satz in der Datei
24	Der Schlüssel des hinzuzufügenden Satzes liegt außerhalb des Schlüsselwertebereiches der Datei
28	Ungenügender virtueller Speicherplatz, um die Abfrage befriedigen zu können
2C	Zu kleines Arbeitsgebiet für das GET-Statement mit OPTCD=MVE
30	Ein VSAM-Verarbeitungsmodul kann wegen ungenügenden Speicherplatzes nicht geladen werden
34	Fehler in VSAM-Routine der IBM (Speicherabzug der IBM übergeben)
38	Fehler beim Zugriff zum VSAM-Katalog; Job wiederholen (bei andauerndem Fehler Speicherabzug an IBM)
40	Anzahl aktiver Abfragen größer als der Wert im STRNO-Operanden
44	Abfrage gilt einer im MACRF-Operanden des ACB nicht vorgesehenen Operation
48	Zugriff mit Schlüssel für eine Datei in Zugangsfolge vorgesehen
4C	Einfügung eines schlüssellosen Satzes in eine Datei in Schlüsselfolge vorgesehen
50	Makro ERASE für eine Datei in Zugangsfolge vorgesehen
54	Makro PUT mit OPTCD=LOC vorgesehen
58	GET-Makro mit OPTCD=SEQ ohne vorhergehende Einstellung gegeben
5C	PUT-Makro mit OPTCD=UPD oder ERASE-Makro ohne vorhergehendes GET-Makro für die Aufdatierung
60	Versuch zur Änderung eines Schlüssels beim Aufdatieren eines Satzes
64	Versuch zur Änderung einer Satzlänge beim Aufdatieren eines Satzes unter Angabe der relativen Byteadresse
68	Ungültige oder widersprüchliche Optionen in der Anforderungsparameterliste
6C	Satzlänge größer als das Maximum oder gleich Null oder kleiner als die Summe aus Schlüssellänge und Schlüsselposition im Satz
70	Schlüssellänge gleich Null oder zu groß
74	PUT-Makro mit den Optionen (OPTCD) DIR, SKP oder UPD während der Erstellung der Datei

Abb. 23.8. Rückmeldewerte bei logischen Fehlern

Im Betriebssystem OS wird außerdem eine 128 Bytes lange Nachricht in das bei MSGAREA genannte Gebiet gestellt, wenn ein physikalischer Fehler aufkommt. Die Nachricht beschreibt den Fehler und ist druckfertig aufbereitet. Wenn der Wert bei MSGLEN kleiner als 128 ist, so landet die Nachricht nicht in dem bei MSGAREA genannten Gebiet.

Die Entscheidung darüber, ob es besser ist, Ausgangsroutinen zu schreiben oder nach jeder Operation den Inhalt des Registers 15 und u. U. anschließend den des Registers 0 zu testen, bleibt dem Programmierer überlassen. In den meisten Fällen ist das unwesentlich, bei längeren Programmen sollte man sich jedoch des Vorteils wegen zur Codierung von Ausgangsroutinen durchringen.

23.8. Die Erstellung von Dateien

Eine VSAM-Datei wird mittels des AMS-Dienstprogrammes[1] generiert. AMS kann auch zur Erstellung der Datei benutzt werden, d.h. daß mit seiner Hilfe zum ersten Mal Sätze in eine Datei hineingestellt werden. Alternativ kann hierzu natürlich auch ein Benutzerprogramm geschrieben werden. Während der Erstellung muß eine Datei in Schlüsselfolge sequentiell mit Zugriffen über die Schlüssel angesprochen werden. Das bedeutet, daß die Satzschlüssel in aufsteigender Reihenfolge vorliegen müssen. Eine Datei in Zugangsfolge muß mit PUT-Makros im sequentiellen Zugriff erstellt werden. Der direkte Zugriff oder der Aufdatierungsmodus kann für die Erstellung einer Datei nicht herangezogen werden.

Wir wollen diese Ausführungen durch ein Beispiel ergänzen. Dazu wollen wir eine Kartendatei lesen, aus der dann die VSAM-Datei erstellt werden soll. Die Karten sind wie folgt abgelocht:

Spalten 1 bis 35: Artikelbezeichnung
Spalten 36 bis 44: Artikelnummer
Spalten 45 bis 51: Einzelpreis
Spalten 52 bis 59: Lagermenge

Der Satz der VSAM-Datei soll folgendes Format aufweisen:

Bytes 0 bis 34: Artikelbezeichnung
Bytes 35 bis 39: Artikelnummer (dezimal gepackt)
Bytes 40 bis 43: Einzelpreis (dezimal gepackt)
Bytes 44 bis 47: Bestandsmenge (dezimal gepackt)

In der Abb. 23.9. liegt ein Programm in der Assemblierersprache für das Betriebssystem DOS, in der Abb. 23.10. ein solches für das Betriebssystem OS vor. Beide Programme sorgen für die Erstellung der VSAM-Datei aus der Kartendatei. Sie gehen von der Annahme aus, daß die Lochkarten nach aufsteigender Artikelnummer geordnet sind; diese ist nämlich gleich dem Schlüssel der VSAM-Datei. Man beachte, daß solche Informationen wie die Stellung des Schlüsselfeldes innerhalb der Sätze nicht im Programm erscheinen. Derartige Fakten sind Teil des VSAM-Kataloges, der bei der Genierung der Datei mittels des Dienstprogrammes AMS entstanden ist. Bei der Eröffnung der Datei, wandern diese Informationen aus dem Katalog in die Felder des Zugriffssteuerblockes ACB.

1) AMS ist die Abkürzung von „Acess Method Services" (Zugriffsmethodendienste)

Die DOS- und OS-Versionen gleichen sich im wesentlichen. Nach dem PUT-Statement machen wir keine Fehlerprüfung, d.h., daß bei Zustandekommen eines Fehlers, z. B. durch Nichteinhaltung der Kartenreihenfolge, der Satz nicht in die Datei aufgenommen wird und ein solcher Fehler auch nicht registriert wird.

DOS

```
DVSI     CSECT
         BALR  11,0
         USING *,11
         LA    13,SAVEAREA
         OPEN  SYSIN,I
DVSL     GET   SYSIN,N
         MVC   P(35),N
         PACK  P+35(5),N+35(9)
         PACK  P+40(4),N+44(7)
         PACK  P+44(4),N+51(7)
         PUT   RPL=RI
         B     DVSL
DVSZ     CLOSE SYSIN,I
         EOJ
SAVEAREA DC    18F'0'
SYSIN    DTFCD BLKSIZE=80,EOFADDR=DVSZ,     1
               WORKA=YES,DEVADDR=SYSIPT,    2
               IOAREA1=IO,DEVICE=3505
I        ACB   MACRF=(KEY,SEQ,OUT)
RI       RPL   OPTCD=(KEY,SEQ,NUP,MVE),     1
               ACB=I,AREALEN=48,            2
               AREA=P,RECLEN=48
N        DS    CL80
P        DS    CL48
IO       DS    CL80
         END   DVSI
```

Abb. 23.9. Beispiel für die Erstellung einer VSAM-Datei im Betriebssystem DOS

Einige Bemerkungen zu den vorliegenden Programmen sollen an dieser Stelle gemacht werden: Das Rettungsgebiet ist mit SAVEAREA bezeichnet, das Kartenarbeitsgebiet mit N und das Arbeitsgebiet für die VSAM-Datei mit P. Das EA-Gebiet der Kartendatei heißt in der DOS-Fassung IO.

Es ist selbstverständlich einleuchtend, daß man bei den meisten kommerziellen Anwendungen Prüfungen auf Fehler und ihre Behandlung einbauen muß; meist wird bei in einem PUT-Statement auftauchenden Fehlern eine Fehlernachricht ausgegeben. Im ACB-Makro müssen wir beim Operanden MACRF die Angabe OUT machen, da diese keine Standardangabe ist. Für die anderen Operanden benutzen wir in der Regel deren Standards. Wir können deshalb auch den Operanden OPTCD beim Makro RPL weglassen, da wir nur dessen Standardwerte bedürfen.

Bei der DOS-Version haben wir ebenfalls ein Rettungsgebiet angelegt; wegen VSAM muß das geschehen. Es ist nicht notwendig, sie mit Anfangswerten zu versorgen. Sehen wir jedoch eine solche Maßnahme vor, so kann uns diese helfen, ein Programm zu testen, da sie uns wesentlich einfacher den Inhalt der verschiedenen Gebiete verstehen hilft, wenn das Programm mit einem unerwarteten Zusammenbruch endet.

OS

```
VSI      CSECT
         USING *,11
         SAVE  (14,12)
         LR    11,15
         LA    14,SAVEAREA
         ST    14,8(13)
         ST    13,4(14)
         LR    13,14
         OPEN  (SYSIN,,I)
VSIL     GET   SYSIN,N
         MVC   P(35),N
         PACK  P+35(5),N+35(9)
         PACK  P+40(4),N+44(7)
         PACK  P+44(4),N+51(7)
         PUT   RPL=RI
         B     VSIL
VSIZ     CLOSE (SYSIN,,I)
         L     13,4(13)
         RETURN (14,12),RC=0
SYSIN    DCB   DSORG=PS,DDNAME=SYSIN,      ¬
               MACRF=GM,EODAD=VSIZ
I        ACB   MACRF=(KEY,SEQ,OUT)
RI       RPL   OPTCD=(KEY,SEQ,NUP,MVE),    1
               ACB=I,AREALEN=48,           2
               AREA=P,RECLEN=48
         DS    0D
N        DS    CL80
P        DS    CL48
SAVEAREA DC    18F'0'
         END   VSI
```

Abb. 23.10. Beispiel für die Erstellung einer VSAM-Datei im Betriebssystem OS

23.9. Betrachtung eines Beispieles

In Abb. 23.11. wollen wir die Codierung eines VSAM-Beispieles verfolgen, bei dem eine Bestelldatei (Dateiname T) benutzt wird, um die Artikeldatei I (siehe Abb. 23.9. bzw. Abb. 23.10.) auf dem neuesten Stand zu halten. Gleichzeitig wird eine zweite Datei C, die Kundendatei, aufdatiert.

Die Sätze der Kundendatei sind wie folgt aufgebaut:

Bytes	Inhalt	Feldname
0 bis 2	Kundennummer (dezimal gepackt)	WCCUST
3 bis 32	Kundenname	WCNAME
33 bis 62	Kundenadresse (Zeile 1)	WCADDR1
63 bis 92	Kundenadresse (Zeile 2)	WCADDR2
93 bis 122	Kundenadresse (Zeile 3)	WCADDR3
123 bis 127	Kredit (dezimal gepackt)	WCBALN

An diesen Stammteil des Satzes kann zwischen 0 und 82 mal eine Feldgruppe angehängt werden. Jede solche Gruppe besteht aus den folgenden Feldern:

Bytes	Inhalt	Feldname
0 bis 34	Artikelbezeichnung	WCDESCR
35 bis 39	Artikelnummer (dezimal gepackt)	WCCODE
40 bis 43	Einzelpreis (dezimal gepackt)	WCPRICE
44 bis 47	Menge (dezimal gepackt)	WCQTY

Das Aufdatierungsprogramm liest die Sätze der Bestelldatei, die wie folgt zusammengestellt sind:

Bytes	Inhalt	Feldname (DSECT)
0 bis 4	Kundennummer (dezimal mit Zonen)	WTCUST
5 bis 99	Menge (dezimal mit Zonen)	WTQTY
10 bis 18	Artikelnummer (dezimal mit Zonen)	WTCODE

Nach dem Lesen jedes Bestellsatzes wird sein Inhalt dazu benutzt, um den entsprechenden Kundensatz um genau 48 Bytes zu erweitern. Dieses Anhängsel wird dabei aus der Artikelnummer und der Menge aus der Bestelldatei sowie aus der Artikelbezeichnung und dem Preis aus der Artikeldatei zusammengesetzt. Kann die Artikelnummer nicht in der Artikeldatei gefunden werden, so wird der Preis auf Null gesetzt und als Bezeichnung wird „INVALID CODE" aufgenommen. Stimmt die Kundennummer in der Bestelldatei mit keiner Eintragung in der Kundendatei überein, so wird der Bestellsatz einfach ignoriert.

Die Kundendatei wird ebenfalls auf den neuesten Stand gebracht, indem vom Kreditbetrag der Gesamtpreis (Menge mal Einzelpreis) subtrahiert wird. Die Artikeldatei selbst wird auch aufdatiert, indem von der Bestandsmenge die Bestellmenge subtrahiert wird.

Die Kundendatei stellt eine Datei in Schlüsselfolge dar; als Schlüssel fungiert die Kundennummer. Die Bestelldatei wird durch eine Datei in Zugangsfolge repräsentiert und enthält Bestellungen in einer beliebigen Reihenfolge.

Das Programm ist recht einfach gehalten. Wir prüfen bei der Bestelldatei nichts, obwohl ein physikalischer Fehler auftreten kann. Beim Lesen des Kundensatzes führt uns ein physikalischer Fehler auf die Stelle VS900, wo wir das Programm beenden. Die einzige andere anormale Bedingung, die im Programm behandelt wird, ist „kein Satz gefunden"; in diesem Fall holen wir einfach den nächsten Bestellsatz. Die beiden gleichen genannten Fehler können sich beim Zugriff zu einem Artikelsatz ereignen. In diesem Fall versucht jedoch die Bedingung „kein Satz gefunden", daß ein Scheinteilsatz erzeugt wird. Außerdem wird eine Kennzeichnung WKI gesetzt, durch die verhindert wird, daß der Satz später aufdatiert wird. Das Makro SHOWCB dient dazu, die Länge des Kundensatzes zu ermitteln; MODCB wird zur Addition von 48 auf die bisherige Länge benutzt. Ist WKI gleich 0, so wird die Aufdatierung der Artikeldatei übergangen.

Die Benutzung des EQU-Statements für die Register soll die Testarbeit erleichtern. Damit erreichen wir, daß in der Referenztabelle, die vom Assemblierer ausgegeben wird, alle Registerreferenzen aufgelistet sind. Bei vielen Installationen gilt diese Technik als selbstverständlich für alle benutzten Register; eine Ausnahme ist nur für die trivialsten Programme zulässig.

Aufgabengruppe 63

1) Schreibe ein Programm, mit deren Hilfe Lochkarten gelesen werden, bei denen Bestellinformationen auf den Spalten 1 bis 19 abgelocht sind und generiere aus ihnen die im Abschnitt 23.9. beschriebene Bestelldatei.

OS

```
VS      CSECT
        USING *,10,11
        SAVE  (14,12)
        LR    10,15
        LA    11,2048
        LA    11,2048(10,11)
        LA    14,SAVEAREA
        ST    14,8(13)
        ST    13,4(14)
        LR    13,14
        OPEN  (C,,T,,I)
        B     VS200
```

DOS

```
VS      CSECT
        BALR  10,0
        USING *,10,11
        LA    11,2048
        LA    11,2048(10,11)
        LA    13,SAVEAREA
        OPEN  C,T,I
        B     VS200
```

```
SAVEAREA DC    18F'0'
VS200    GET   RPL=RT
         L     R3,WKLENT
         USING WT,R3
         PACK  WKCUST,WTCUST
         PACK  WKCODE,WTCODE
         PACK  WKQTY,WTQTY
VS300    GET   RPL=RC
         B     VS320(15)
VS320    B     VS400      X'00'
         B     VS900      X'04'
         B     VS200      X'08'
         B     VS900      X'0C'
VS400    GET   RPL=RI
         B     VS420(15)
VS420    B     VS600      X'00'
         B     VS900      X'04'
         B     VS500      X'08'
         B     VS900      X'0C'
VS500    MVI   WKI,X'00'
         MVC   WIDESCR,=CL35'INVALID CODE'
         MVC   WIPRICE,=PL4'0'
         MVC   WIQTY,=PL4'0'
         MVC   WICODE,WKCODE
VS600    SHOWCB RPL=RC,FIELDS=(RECLEN),  ¬
               AREA=WKLENC,LENGTH=4
         L     R4,WKLENC
         LA    R5,WC(R4)
         MVC   0(44,R5),WI
         MVC   44(4,R5),WKQTY
         ZAP   WKMPY,WKQTY
         MP    WKMPY,WIPRICE
         SP    WCBALN,WKMPY
         LA    R6,48(R4)
         MODCB RPL=RC,RECLEN=(R6)
         PUT   RPL=RC
         SP    WIQTY,WKQTY
         TS    WKI
         BZ    VS200
         PUT   RPL=RI
         B     VS200
```

Abb. 23.11. Beispiel für ein Programm zum Aufdatieren verschiedener Dateien (Teil 1)

OS

```
VS900    CLOSE (C,,T,,I)
         L     13,4(13)
         RETURN (14,12),RC=0
           *                CONTROL BLOCKS
           C        ACB     MACRF=(KEY,DIR,OUT)
           I        ACB     MACRF=(KEY,DIR,OUT)
           T        ACB     MACRF=(ADR,SEQ,IN),EXLST=X
           X        EXLST   EODAD=(VS900,A)
           RC       RPL     OPTCD=(KEY,DIR,UPD,MVE),    1
                            ACB=C,AREALEN=4064,         2
                            AREA=WC,ARG=WKCUST
           RI       RPL     OPTCD=(KEY,DIR,UPD,MVE),    1
                            ACB=I,AREALEN=48,           2
                            AREA=WI,ARG=WKCODE
           RT       RPL     OPTCD=(ADR,SEQ,NUP,LOC),    1
                            ACB=T,AREALEN=4,            2
                            AREA=WKLENT
                    LTORG
           *                WORK AREAS
           WKLENC   DS      F
           WKLENT   DS      F
           WKCUST   DS      PL3
           WKCODE   DS      PL5
           WKQTY    DS      PL4
           WKI      DC      X'FF'
           WKMPY    DS      PL8
           *                VSAM AREAS
                    DS      0D
           WI       DS      0CL48
           WIDESCR  DS      CL35
           WICODE   DS      PL5
           WIPRICE  DS      PL4
           WIQTY    DS      PL4
           WC       DS      0CL4064
           WCCUST   DS      PL3
           WCNAME   DS      CL30
           WCADDR1  DS      CL30
           WCADDR2  DS      CL30
           WCADDR3  DS      CL30
           WCBALN   DS      PL5
           WCITEM   DS      81CL48
           WCDESCR  DS      CL35
           WCCODE   DS      PL5
           WCPRICE  DS      PL4
           WCQTY    DS      PL4
           WT       DSECT
           WTCUST   DS      CL5
           WTQTY    DS      CL5
           WTCODE   DS      CL9
           R3       EQU     3
           R4       EQU     4
           R5       EQU     5
           R6       EQU     6
                    END     VS
```

DOS

```
VS900    CLOSE C,T,I
         EOJ
```

Anmerkungen:

1) Der Kommentar WORK AREAS leitet die Definitionen von Arbeitsgebieten ein.
2) Der Kommentar VSAM AREAS leitet die Definitionen von Gebieten ein, die für VSAM benötigt werden.
3) Der Kommentar CONTROL BLOCKS steht an der Spitze der Steuerblockfestlegungen.
4) Das Rettungsgebiet ist mit SAVEAREA bezeichnet.

Abb. 23.11. Beispiel für ein Programm zum Aufdatieren verschiedener Dateien (Teil 2)

2) Ändere das Programm von Abb. 23.11. wie folgt ab: Es ist nach VS700 zu verzweigen, wo ein neuer Kundensatz anzulegen ist. Für den Namen ist „NEUNAME", für die Adresse sind Leerzeichen und für den Kredit Null einzusetzen. Baue diesen Satz auf, verarbeite ihn und füge ihn in die Datei ein. Benutze das Makro MODCB und die gleiche Anforderungsparameterliste wie für die Aufdatierung.
3) Löse die Aufgabe 2) unter Benutzung einer getrennten Anforderungsparameterliste namens RA für das Hinzufügen eines Satzes!
4) Erweitere die Codierung des Programmes von Abb. 23.11. wie folgt:
   a) Bei VS900 ist ein

```
TCLOSE
```

   bzw.

```
CLOSE TYPE=T
```

   zu codieren. Nach einem geeigneten POINT-Makro und anderen erforderlichen Makros ist die Kundendatei sequentiell zu lesen.
   b) Drucke außerdem den gesamten Inhalt derjenigen Kundensätze mit dem Namen „NEUNAME" und leeren Adreßfeldern.
5) Schreibe ein Programm, daß die Artikeldatei (Abb. 23.9. bzw. 23.10.) verarbeitet, wobei Karten als Eingabe dienen. Durch eine Kennung auf Spalte 80 sind die verschiedenen Kartenarten und damit die aus ihnen resultierende Verarbeitung festgelegt:
   - Kartenart 2
     Hinzufügung eines neuen Artikelsatzes
   - Kartenart 3
     Die auf solchen Karten verzeichnete Menge ist der Bestandsmenge hinzuzufügen
   - Kartenart 4
     Die auf solchen Karten verzeichnete Menge ersetzt die Bestandsmenge
   - Kartenart 5
     Entfernen des Artikelsatzes
   - Kartenart 6
     Drucken des Artikelsatzes

Es kann vorausgesetzt werden, daß die Karten nach aufsteigenden Artikelnummern sortiert sind; es ist deshalb die sequentielle Verarbeitung mit Überspringen (SKIP) vorzusehen.

24. Verschiedenes

24.1. Arbeitsgruppen mit Chefprogrammierern

In großen Computerinstallationen liegt der Arbeitsdurchschnitt der Programmierer bei einer ausgetesteten Instruktion pro Stunde[1]. Die Fähigkeit großer Computer, mit vielen umfangreichen Programmen gleichzeitig beschäftigt zu sein, führte zur Entwicklung vieler Programme, die Hunderte, ja sogar Tausende von Instruktionen umfassen. Für zu viele dieser Programme führt die Testphase zur Einführung neuer Fehler in fast dem gleichen Umfang wie alte ausgemerzt werden.

Einen Schritt vorwärts zur Entwicklung großer Programme nach Plan und mit vertretbaren Kosten liegt in einer geeigneten Organisation des Programmieraufwandes. Die meiste Zeit benötigt ein Programmierer zum Testen. Seine zweithäufigste Aktivität liegt in der Verbesserung seiner Programme aufgrund fehlerhafter Logik oder ungenügender Absprache mit anderen Programmierern. Ein organisatorischer Fortschritt für das Schreiben umfangreicher Programme liegt sicherlich in der Bildung von Arbeitsgruppen unter Leitung von Chefprogrammierern.

Ein solches Team setzt sich aus einem Chefprogrammierer, einem Erstprogrammierer, einem Sekretär, etwa vier Juniorprogrammierern und im Bedarfsfall aus einer Verwaltungskraft zusammen.

Der Chefprogrammierer ist ein versierter, erfahrener Programmierer, der gewissermaßen als technischer Manager des Programmierteams anzusehen ist. Er ist sowohl für den gesamten Programmentwurf als auch für den Systementwurf verantwortlich. Er codiert die kritischen Segmente selbst, an denen die weniger erfahrenen Programmierer sonst scheitern würden. Er legt fest, welche Programme jedes Mitglied des Programmierteams zu schreiben hat, er überprüft die Programme der anderen Mitarbeiter und er überwacht schließlich die Einfügung jedes einzelnen Programmes ins System.

Der Erstprogrammierer muß ebenfalls mit dem kompletten System vertraut sein. Er nimmt an allen wesentlichen Entscheidungsprozessen teil. Er muß in der Lage sein, die Führung des Teams immer dann zu übernehmen, wenn es notwendig ist. Er hilft mit beim Systementwurf und er codiert die Schlüsselteile des Systems. Er wirkt somit gewissermaßen als rechte Hand des Chefprogrammierers. Außerdem liegt in seiner Obhut die Planung des Systemtests inklusive der Bereitstellung der Testdaten.

Der Sekretär oder die Sekretärin beschäftigt sich mit den Projektberichten einschließlich der Programmlisten und der maschinenlesbaren Kopien aller Programme. Er oder sie verwaltet die Resultate aller Testläufe und faßt alle Projektberichte lose in Ordnern zusammen. Programmablaufpläne und andere Dokumentationen werden von ihm oder ihr geschrieben und er bzw. sie hält dieses Material bei Änderungen auch auf dem laufenden. Kurz und gut, seine bzw. ihre Funktion ist die, die Programmierer von allen Arbeiten zu entbinden, die nichts mit dem Programmieren zu tun haben.

Unter Aufsicht beschäftigen sich die Juniorprogrammierer mit den detaillierten Programmablaufplänen, mit dem Codieren, Testen und der Vorbereitung von Testdaten für ihre Programme. Jedem Programmierer sind wenige Quellenmoduln zugeteilt, die er zu schreiben hat. Gewöhnlich legt der Wartungsprogrammierer die Testausrüstung fest, unter der jedes Programm erst laufen muß, bevor es ins Gesamtsystem integriert wird.

---

1) Im übrigen werden in der Literatur unterschiedliche Zeiten veröffentlicht (Anmerkung von R. Gritsch).

Die Verwaltungskraft, die dem Team zugeordnet ist, beschäftigt sich in der Regel mit allen administrativen, finanziellen Problemen, überhaupt mit allen Problemen, die nichts mit dem Programmieren zu tun haben. Bei manchen Projekten werden natürlich solche Kräfte nicht benötigt.

Der Vorteil eines solchen Teams liegt hauptsächlich darin, daß die überragenden Fähigkeiten einer Spitzenkraft wie die des Chefprogrammierers ausgenutzt werden, um einmal den Erstprogrammierer so weit zu fördern, daß er es ebenfalls zum Chefprogrammierer bringt, zum anderen, daß er solche Dinge von den anderen Programmierern wegnimmt, zu deren Erledigung sie noch keine ausreichende Erfahrung besitzen.

Der Nutzen eines Teams zeigt sich auch darin, daß der Chefprogrammierer eine detaillierte Kontrolle über alle Programmieranstrengungen ausüben kann. Dadurch wird Doppelarbeit weitgehend vermieden und Kommunikationsschwierigkeiten werden auf ein Mindestmaß reduziert. Die Tätigkeit der Sekretärin ermöglicht die ständige komplette Übersicht über die historische Entwicklung des Projektes und sichert überdies, daß jeder Mitarbeiter jederzeit Zugriff zu den wichtigsten Dokumentationsunterlagen hat. In geeigneter Weise benutzt, bewahren die Berichte der Sekretärin das Team davor, daß sich ein Programmierer Programmänderungen auferlegt, die er wegen Unkenntnis des letzten Entwicklungsstandes sonst vielleicht überflüssigerweise in Angriff genommen hätte.

24.2. Strukturierte Programmierung

Eine andere Idee zur Erzielung besserer Programme, die noch dazu schnell und mit weniger Fehlern geschrieben sind, fußt auf der strukturierten Programmierung. Gewöhnlich kristalliert sich bei Realisierung dieser Idee auch eine bessere Dokumentation heraus. Es ist nämlich möglich, ein Programm mit nur drei Steuerstrukturen zu schreiben. Sie sind unter a), b) und c) in Abb. 24.1. dargestellt. Die Benutzung solcher Strukturen zeigt, daß absolute Verzweigungen nicht notwendig bei der Programmierung sind. Die Eliminierung der Sprünge führt zu Programmen, deren Logik viel einfacher verfolgt werden kann. Da die Programmlogik das Hauptabschreckungsmittel beim Testen ist, kommt es dadurch von allein zu besseren Programmen, die zudem wesentlich übersichtlicher sind.

In der Praxis erlauben wir für gewöhnlich fünf Typen von Steuersequenzen; Abb. 24.1. verweist auf sie. (Bei der Schleife macht es dabei keinen Unterschied, ob wir den Entscheidungsblock vor den Verarbeitungsblock stellen oder umgekehrt). Die Mehrfachverzweigung (Verteiler) ist im Grunde genommen nichts weiter als eine Ausdehnung bedingter Verzweigungen. Als wichtigstes Argument für Strukturen spricht, das sie nur einen Eingang und einen Ausgang besitzen. Die Schleife mit einem anormalen Ausgang stellt eine Verletzung des Konzeptes für die strukturierte Programmierung dar, nämlich, daß jede Struktur nur einen Eingang und einen Ausgang besitzt. Natürlich kann der anormale Ausgang viel Programmablaufzeit einsparen helfen. Das wiederum kann wichtig bei der Behandlung unerwarteter Programmunterbrechungen sein.

Der zweite bei der strukturierten Programmierung zu beachtende Aspekt ist die Strukturlänge. Jeder Programm-Modul setzt sich aus einem oder mehreren Segmenten zusammen, die in gewünschter Weise miteinander verbunden sind. Jeder Modul soll sich mit einer einzigen Sache befassen und nicht mit einer Sache plus Teilen von anderen Sachen. Wird eine problemorientierte Sprache benutzt, so sollte jeder Modul auf nicht mehr als auf etwa 50 Statements begrenzt sein. In der AL sollte ein Maximum von 75 bis 100 Statements nicht überschritten werden. Die längsten Moduln sind gewöhnlich dabei

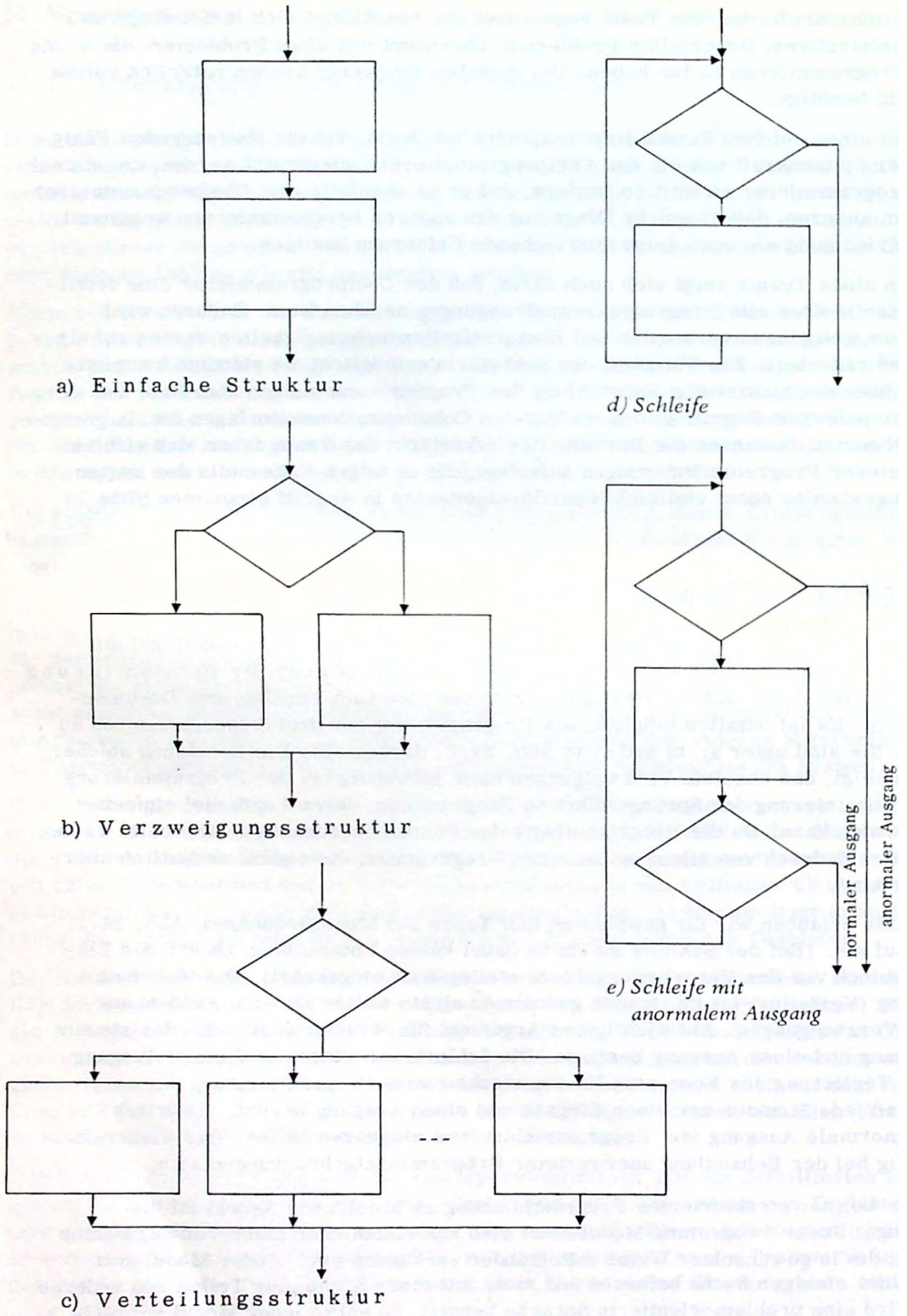


Abb. 24.1. Programmstrukturen

diejenigen, die einen Eingabesatz auf gültige Verschlüsselungen überprüfen oder die Felder aufbereiten und zum Drucken in eine Zeile hineinstellen. Überschreitet ein Quellenmodul etwa 50 Statements, so ist es unwahrscheinlich, daß dieser Modul nur eine einzige Funktion erledigt und sie überdies in einer verständlichen Weise ausführt.

Da PL/1 selbst geradezu wunderbar zur strukturierten Programmierung hinführt, wird es im weitesten Maße für die Schaffung strukturierter Programme eingesetzt.

Programmiererteams mit Chefprogrammierern haben inzwischen Codierungsraten von fünf Instruktionen pro Programmiererstunde erreicht, wobei die Fehlerhäufigkeit bei ausgetesteten und freigegebenen Programmen auf weniger als zwei Fehler pro Arbeitsjahr eines Programmierers herabgesunken ist.

Die strukturierte Programmierung erfordert allgemein die Entwicklung sorgfältig überlegter Programmablaufpläne im groben, denen sich etwas verfeinerte anschließen sollten. Solche verfeinerten sollen nur so weit detailliert werden, daß sie gerade jeden zu codierenden Modul als gesonderten Verarbeitungsblock ausweisen.

24.3. Rekursive Programmierung

Ein Lademodul heißt r e e n t r a n t (jederzeit wiederbetretbar), wenn er erneut aktiviert werden kann, obwohl er für eine andere Task (Programm) noch aktiv ist. Ein Lademodul heißt r e k u r s i v, wenn er erneut für die gleiche Task aktiviert werden kann. Ein Lademodul ist s e r i e l l w i e d e r v e r w e n d b a r, wenn er im nichtaktiven Zustand reaktiviert werden kann, nachdem er zuvor schon ausgeführt worden ist.

Für ein seriell wiederverwendbares Programm ist es erforderlich, daß die im Modul vorkommenden Konstanten entweder nicht geändert werden oder aber jedes Mal nach Benutzung des Moduls auf ihre ursprünglichen Werte zurückgeführt werden. Unterprogramme werden in der Regel so geschrieben; für die meisten Hauptprogramme kann man diese Feststellung nicht treffen. PL/1-Programme initialisieren die automatische Speicherklasse jedes Mal, wenn ein Modul ausgeführt wird, sie unterdrücken diese Maßnahme jedoch für die statische Speicherklasse. Wenn also die Inhalte von Teilen der statischen Speicherklasse während der Programmausführung geändert und die Änderungen nicht wieder rückgängig gemacht werden, so ist ein solcher Modul nicht wiederverwendbar im seriellen Sinne.

Programme, die jederzeit wiederbetretbar sind, werden in der Regel für Supervisoraktivitäten geschrieben. Nehmen wir z. B. einmal an, daß Programm A gerade in dem Augenblick den IOCS-Modul B ausführen läßt, in dem eine Unterbrechung von einem Programm höherer Priorität durchkommt. Dieses bekommt die Programmablaufsteuerung zugeteilt und versucht nun selbst, den IOCS-Modul B auszuführen, obwohl gerade A vielleicht gerade erst bis zur Mitte dieses Moduls vorgedrungen ist. Wenn B reentrant geschrieben ist, so macht das nichts aus. — Das Schreiben von jederzeit wiederbetretbaren Programmen erfordert es, daß alle während der Ausführung zu modifizierenden Felder in Registern stehen oder relativ zu Registern adressiert werden, deren Werte abhängig von den Informationen sind, die an diese Routine gehen. Für Anwendungsprogrammierer besteht selten ein Grund, jederzeit wiederbetretbare Programme zu schreiben.

Rekursive Programmierung bedeutet, daß nach Aufruf des Unterprogrammes D vom Programm C aus dieses sich während seiner Ausführung selbst erneut aufrufen kann. Z. B. ist es also damit möglich, daß D das Unterprogramm E aufruft, das selbst wiederum D benötigt. Es ist ein wenig leichter, rekursive als jederzeit wiederbetretbare

Programme zu schreiben, da bei der rekursiven Programmierung der Programmierer die Bedingungen selbst kontrollieren kam, unter denen sich sein Modul selbst aufrufen kann.

Abb. 24.2. zeigt ein in PL/1 geschriebenes Hauptprogramm, daß das in Abb. 24.3. dargestellte rekursive Programm aufruft. Das Hauptprogramm liest eine Lochkarte, die eine Dezimalzahl x enthält, die in eine ein Vollwort belegende Dualzahl konvertiert wird. Das Hauptprogramm ruft DF mit zwei Argumenten, nämlich X und D auf. D ist eine Variable, deren Wertevorrat aus Dualzahlen besteht, deren Stellenzahl gerade ein Vollwort umfaßt; für D gibt es keinen Anfangswert. Das Unterprogramm berechnet

$$D = X!!$$

Dabei ist die Doppelfakultät wie folgt definiert:

$$n!! = n\,(n-2)\,(n-4)\,(n-6)\cdots 2$$
$$\text{oder} \quad n!! = n\,(n-2)\,(n-4)\,(n-6)\cdots 1$$

Einige Beispiele hierzu mögen folgen:

a) $5!! = 5 \cdot 3 \cdot 1 = 15$
b) $7!! = 7 \cdot 5 \cdot 3 \cdot 1 = 105$
c) $12!! = 12 \cdot 10 \cdot 8 \cdot 6 \cdot 4 \cdot 2 = 46080$

Wir gestehen gern zu, daß es einfachere Wege gibt, den Funktionswert zu berechnen, aber es handelt sich hier um ein einfaches Beispiel, das ausreichend illustriert, was wir zu beachten haben.

```
BSP24:  PROCEDURE OPTIONS(MAIN) ;
          DECLARE  (X,D) BINARY FIXED(31) ;
          ON ENDFILE(SYSIN)  GOTO E ;
EIN:      GET LIST (X) ;
          CALL DF(X,D) ;
          PUT SKIP LIST (X,D) ;
          GOTO EIN ;
E:      END ;
```

Abb. 24.2. Hauptprogramm zum Aufruf des rekursiven Unterprogrammes DF

Das in der AL geschriebene Unterprogramm ist in einer für das Betriebssystem OS gültigen Fassung dargestellt. Seine Übertragung auf das Betriebssystem DOS erfordert folgende Änderungen:

OS	DOS	
GETMAIN	GETVIS	(ohne Operanden)
FREEMAIN	FREEVIS	(ohne Operanden)

Man beachte, daß bei jedem Aufruf des Unterprogrammes ein neues Rettungsgebiet angelegt wird. Dieses umfaßt im OS 80 Bytes, im DOS dagegen wegen der durch das System veranlaßten Rundung auf Vielfache von 128 etwas mehr, nämlich 128 Bytes. Das Rettungsgebiet nimmt nicht nur wie üblich die Registerstände etc. (72 Bytes) auf, sondern sieht auch Platz für Y vor, einer Zwischenvariablen, die benötigt wird, wenn sich das Programm selbst aufruft.

Verfolgen wir die Schritte einmal, die für die Berechnung von 5!! ablaufen:

1) Das Hauptprogramm ruft DF mit den Argumenten 5 und D auf, wobei D irgendeinen Wert besitzen darf.

```
         ISEQ  76,80                                                  24030001
DF       CSECT                                                        24030002
         USING *,11                                                   24030003
         SAVE  (14,12)                                                24030004
         LR    11,15                                                  24030005
         LR    2,1        (2): A(A(X),A(D))                           24030006
         LA    0,80                                                   24030007
         GETMAIN R,LV=(0)                                             24030008
         ST    13,4(1)                                                24030009
         ST    1,8(13)                                                24030010
         LR    13,1       (3):  A(X)                                  24030011
         LM    3,4,0(2)   (4):  A(D)                                  24030012
         L     5,0(3)     (5):  X                                     24030013
         LA    6,76(13)   (6):  A(Y)                                  24030014
         ST    5,0(4)     SETZEN VON D GLEICH X                       24030015
         S     5,ZWEI                                                 24030016
         ST    5,0(6)     SETZEN VON Y AUF X-2                        24030017
         BNP   DF3        AUSGANG BEI X KLEINER ALS 3                 24030018
         CALL  DF,((6),(4))                                           24030019
DF2      L     7,0(4)     (7):  D                                     24030020
         A     5,ZWEI     (5):  X=Y+2                                 24030021
         MR    6,5        (7):  D*X                                   24030022
         ST    7,0(4)     SETZEN VON D AUF DAS PRODUKT VON D*X        24030023
DF3      LR    1,13                                                   24030024
         LA    0,80                                                   24030025
         L     13,4(13)                                               24030026
         FREEMAIN R,LV=(0),A=(1)                                      24030027
         RETURN (14,12)                                               24030028
ZWEI     DC    F'2'                                                   24030029
         END                                                          24030030
```

Abb. 24.3. Rekursives Unterprogramm DF zur Berechnung der Doppelfakultät

2) In DF wird D auf 5 und Y auf 3 gesetzt; danach erfolgt der Aufruf von DF mit den Argumenten 3 und D.

3) In DF wird D auf 3 und Y auf 1 gesetzt; danach erfolgt der Aufruf von DF mit den Argumenten 1 und D.

4) In DF wird D auf 1 und Y auf -1 gesetzt; danach erfolgt eine Verzweigung nach DF3, von wo aus die Rückkehr zum zweiten Aufruf erfolgt.

5) Im Rückkehrpunkt DF2 erfolgt die Multiplikation von D (gleich 1) mit Y+2 (gleich 3). Das gebildete Produkt 3 wandert nach D. Es erfolgt anschließend die Rückkehr zum ersten Aufruf von DF.

6) An der Rückkehrstelle DF2 wird die Multiplikation von D (gleich 3) mit Y+2 (gleich 5) veranlaßt. Das Resultat ($3 \cdot 5 = 15$) wird D zugewiesen. Anschließend erfolgt die Rückkehr ins Hauptprogramm.

Der Wertevorrat des Argumentes X, der von diesem Programm beherrscht wird, ist auf den größten Wert beschränkt, der keinen Überlauf zur Folge hat, wenn die Multiplikationen durchgeführt werden. Eine zweite Beschränkung ergibt sich aus der Zahl der Rettungsgebiete, die in der Benutzerpartition untergebracht werden können. Es ist noch anzumerken, daß unser Programm auf jede Überlaufprüfung verzichtet.

24.4. Die Definitionen in der Makrosprache

Der Assemblierer für das System IBM /370 erlaubt es dem Benutzer, daß er seine eigenen Makros definiert. Diese können der Makrobibliothek des Systems hinzugefügt und von jedem in der AL[1] geschriebenen Programm benutzt werden. Alternativ kann eine Makrodefinition auch zu Beginn eines in der AL geschriebenen Quellenmoduls vorgenommen werden. In diesem Fall steht das Makro nur für den betreffenden Modul zur Verfügung. Eine vollständige Diskussion der Makrodefinitionen würde den Rahmen dieses Buches sprengen; deshalb verzichten wir darauf. Wir illustrieren das ganze durch ein Beispiel.

In Abb. 24.4. wird das Makro DPAS gezeigt, das arithmetische Operationen doppelter Genauigkeit mit Dualzahlen in Festpunktdarstellung bewerkstelligt. Genauer gesagt: wenn wir

```
        DPAS  ARD,KLT,PER,WEHY
```

schreiben, so werden aus DPAS Instruktionen erzeugt, die zur Berechnung des Ausdrucks ARD+KLT-PER führen, wobei das Resultat nach WEHY gespeichert wird. Alle Variablen belegen zwei Vollwörter.

```
        MACRO                                          2404000
&N      DPAS  &W,&X,&Y,&Z                              2404000
        AIF   ('&W' EQ '').L1                          2404000
&N      LM    14,15,&W                                 2404000
        AGO   .L2                                      2404000
.L1     ANOP                                           2404000
&N      SR    14,14                                    2404000
        LR    15,14                                    2404000
.L2     AIF   ('&X' EQ '').L3                          2404000
        AL    15,&X+4                                  2404001
        BC    12,*+8                                   2404001
        A     14,=F'1'                                 2404001
        A     14,&X                                    2404001
.L3     AIF   ('&Y' EQ '').L4                          2404001
        SL    15,&Y+4                                  2404001
        BC    3,*+8                                    2404001
        S     14,=F'1'                                 2404001
        S     14,&Y                                    2404001
.L4     AIF   ('&Z' EQ '').L5                          2404001
        STM   14,15,&Z                                 2404002
.L5     MEND                                           2404002
```

Abb. 24.4. Definition des Makros DPAS

Sehen wir uns nun die Statementfolge von Abb. 24.4. an. Das erste Statement MACRO signalisiert dem Assemblierer, daß die Definition eines Makros folgt. Das nächste Statement ist das sogenannte Kopfstatement. Es enthält den Prototyp des Makrostatements. Dabei ist DPAS der Makroname; außerdem ist DPAS von vier Positionsoperanden, jedoch von keinem Schlüsselwortoperanden abhängig. Der Name jeder Scheinvariablen beginnt mit dem &-Zeichen, damit sie von den gewöhnlichen Variablen unterschieden werden kann. Kommt es in einem Programm zur Übernahme des Makros, so werden alle Scheinvariablennamen durch die Namen der Variablen ersetzt, die im Makrostatement an ihre Stelle getreten sind. Wenn wir z. B. das oben aufgeführte Makrostatement codieren, so wird der Code erzeugt, der in Abb. 24.5. vorgestellt ist.

1) AL ist die Abkürzung von „Assembler Language" (Assemblierersprache)

```
LM      14,15,ARD
AL      15,KLT+4
BC      12,*+8
A       14,=F'1'
A       14,KLT
SL      15,PER+4
BC      3,*+8
S       14,=F'1'
S       14,PER
STM     14,15,WEHY
```

Abb. 24.5. Codegenerierung für das in Abb. 24.4. definierte Makro

Nun halten wir einige Bemerkungen zur Makrodefinition selbst für angebracht. Das Statement AIF prüft, ob & W gleich '' ist, d. h. gleich einer Nullkette ist. Das ist der Fall, wenn der Operand fehlt, d.h. wenn z.B. das Makro in der Form

```
DPAS    ,KTU,PQH,WEHZ
```

benutzt wird. Fehlt &W, so gehen wir über zum Statement, das mit .L1 markiert ist. Im anderen Fall fahren wir mit dem nächsten Statement fort, d.h. mit der LM-Instruktion. Das AGO-Statement besagt dem Assemblierer, daß er zur Marke .L2 überzuleiten hat, d.h., daß das nächste zu übersetzende Statement bei .L2 zu finden ist. Aus dem ANOP-Statement wird selbst kein Maschinencode erzeugt, es dient einfach als Scheinoperationscode, um den Platz für die Marke .L1 zu schaffen.

Die auf .L1 folgenden beiden Instruktionen werden in die Makroexpansion nur dann einbezogen, wenn der erste Operand fehlt. Sie haben die Löschung der Register 14 und 15 zur Folge, d.h. der erste Operand wird gleich Null gesetzt.

Das Statement AIF mit der Marke .L2 besorgt den Übergang zur Marke .L3, wenn der zweite Operand & X fehlt. Man beachte hierbei die Benutzung des Literals. Es erscheint unter den ersten Literalen, die auf die Makroexpansion zusammengefaßt erscheinen. Das AIF-Statement bei .L3 überspringt die nächsten vier Instruktionen, wenn der dritte Operand &Y fehlt. Und schließlich besorgt das AIF-Statement mit der Marke .L4, daß die Resultatspeicherung übersprungen wird, wenn der letzte Operand vermißt wird. Durch das MEND-Statement wird dem Assemblierer mitgeteilt, daß das Ende der Makrodefinition vorliegt.
Ein bei der Benutzung des Makros DPAS diesem mitgegebene Marke wird entweder zur LM- oder zur SR-Instruktion gestellt, so wie es die Makrodefinition zeigt. Nehmen wir einmal an, daß wir

```
Q       DPAS    ,R,S
```

codiert hätten. Daraus generiert der Assemblierer die in Abb. 24.6. gezeigte Statementfolge und fügt sie an Stelle des Makros ins Programm ein.

Das Resultat wird in den Registern 14 und 15 abgeliefert, wo es beispielsweise für eine nachfolgende Division aufbewahrt werden kann.

Mit der Makrosprache ist die Mehrzahl der Programmierer nicht vertraut. Zwei oder drei Seniorprogrammierer reichen meist aus, um die von einer Installation benötigten Makros zu schreiben und zu pflegen.

```
Q       SR      14,14
        LR      15,14
        AL      15,R+4
        BC      12,*+8
        A       14,=F'1'
        A       14,R
        SL      15,S+4
        BC      3,*+8
        S       14,=F'1'
        S       14,S
```

Abb. 24.6. Generierter Code bei Benutzung des Makros DPAS bei einem Beispiel

24.5. Sortieren

Gehen wir von einem Anwendungsbeispiel aus: Eine Supermarktkette stanzt bei jeder Warenbewegung 26 Zeichen in einen Lochstreifen. Diese schließen Preis, Artikelnummer und die Filialnummer ein. Lochstreifen mit über 150000 Transaktionen fallen an einem Geschäftstag an. Die Lochstreifen werden in den Computer eingelesen. Die Sätze müssen nun nach der Artikelnummer geordnet werden. Nur so können die Sätze für die Lagerhaltung auf den neuesten Stand gebracht werden und automatisch Auslieferungen an diejenigen Filialen veranlaßt werden, bei denen gewisse Artikel ihren Mindestbestand unterschritten haben. Abhängig vom verfügbaren Hauptspeicherplatz, der ZE-Geschwindigkeit und den für die Zwischenspeicherung zur Verfügung stehenden EA-Geräten kann ein solcher Sortiervorgang weniger als zwei Minuten, aber auch mehr als 50 Minuten dauern. Ein anderer wesentlicher Aspekt ist das zum Sortieren benutzte Computerprogramm. Die IBM stellt ihren Kunden mehrere verschiedene Sortierprogramme zur Verfügung. Aufgabe der Systemprogrammierer ist es, die besonderen Bedürfnisse einer Installation festzustellen und aufgrund dieser ein geeignetes Sortierprogramm auszuwählen. Die meisten Installationen begnügen sich freilich mit einem einzigen Sortierprogramm.

Unsere Ausführungen beschränken sich auf die allgemeinen Charakteristiken der am häufigsten benutzten Sortierprogramme für die IBM /370.

Die zum Sortieren benötigten EA-Dateien schließen ein:

1) Ein Steuerkartensatz muß eingegeben werden, auf dem die Felder genannt sind, die dem Sortieren zugrundegelegt werden sollen,
2) eine Eingabedatei, d.h. die zu sortierende Datenmenge,
3) eine Ausgabedatei, d.h. die sortierte Datenmenge,
4) mehrere Arbeitsdateien, die zur zeitweiligen Zwischenspeicherung der Datensätze benötigt werden (ihre Anzahl beträgt zwischen 1 und 32, als Datenträger kommen Magnetbänder oder Plattenspeicher in Frage),
5) wahlweise eine einem Drucker zugewiesene Ausgabedatei, über die Nachrichten über den Sortierverlauf, wie die Zahl der sortierten Sätze, gedruckt werden (denkbar ist hierfür auch die Konsolausgabe).

Die Sortierprogramme sind so gefertigt, daß sie für mehrere Ausgänge präpariert sind. Darunter versteht man Programmstellen, an denen die Programmablaufsteuerung an Routinen übergeben wird, die vom Benutzer zu schreiben sind. Mit ihnen kann ein Satz

nach dem Lesen, jedoch vor dem Sortieren bearbeitet werden. Diese sogenannten Exitroutinen erfordern gewöhnlich besondere Programmieranstrengungen und werden nicht oft benutzt. Zwei Ausgänge sind jedoch von Interesse für uns. Der Ausgang E15 erlaubt, daß ein Benutzerprogramm die zu sortierenden Sätze bereitstellt, d.h. sie rühren nicht von einer Eingabedatenmenge her. Der Ausgang E35 wiederum stellt die sortierten Sätze an ein Benutzerprogramm zurück, verzichtet also auf ihre Einreihung in eine Ausgabedatenmenge.

Unter den Möglichkeiten, die die an Benutzerausgänge angeschlossene Routinen erledigen können, befinden sich solche wie

a) Veränderung der Satzlänge bei Eingabe- oder bei Ausgabedateien,
b) Löschen von Sätzen,
c) Einfügen neuer Sätze,
d) Herstellen von mehr als einer Kopie der sortierten Sätze.

Im allgemeinen läuft ein Sortierprogramm um so schneller ab, je mehr virtueller Speicherplatz ihm zur Verfügung gestellt werden kann. Doch gilt es zu beachten, daß das Sortieren oft zu häufigem Seitenwechsel führen kann, was wiederum die Leistungsfähigkeit leicht herabsetzen kann. Somit ist für ein Sortierprogramm der Idealumfang des benötigten Speicherplatzes nicht leicht zu ermitteln. Die Erfahrung lehrt, daß man bei der Benutzung des Betriebssystems DOS dem Sortierprogramm nicht mehr als die Hälfte des gesamten vorhandenen realen Speicherplatzes an virtuellem Speicherplatz zur Verfügung stellen sollte. Im Betriebssystem OS hat sich bisher erwiesen, daß etwa ein Drittel oder ein Viertel des realen Speicherraumes für Sortierprogramme nahezu ideale Verhältnisse schaffen, keinesfalls sollten aber weniger als 60 k für umfangreiche Sortierläufe zur Verfügung stehen.

Zur Sortierung sind Arbeitsdateien auf Magnetbändern oder Plattenspeichern erforderlich. Bei Benutzung von Bändern müssen wenigstens drei zur Verfügung stehen. Werden Plattenspeicher eingesetzt, so muß der gesamte Speicherplatz auf ihnen mindestens 2,5 mal so groß wie derjenige sein, der zur Aufnahme der gesamten Datenmenge notwendig ist, wenn dabei die Sätze zu Sortierblöcken zusammengefaßt sind. Den Begriff „Sortierblock" können wir nicht exakt definieren. Beim Sortieren wird nämlich versucht, die Sätze zu den größten Blöcken zusammenzufassen, die noch verkraftet werden können, d.h. das Sortierprogramm kann sich keine Einschränkung des Bedarfs an virtuellem Speicherplatz leisten, den es für seine eigenen Zwecke noch benötigt. Je größer die Partition ist, um so größer sind naturgemäß die Sortierblöcke. Werden Arbeitsdateien auf Plattenspeicher gelegt, so ist es ratsam, nicht mehr als eine Arbeitsdatei auf einen Plattenstapel zu legen. Halten wir fest: Gelingt es uns, nur eine Arbeitsdatei pro Stapel einzurichten, so können wir erwarten, daß das Sortieren um so schneller abläuft, je mehr Arbeitsdateien wir präpariert haben. Alle Arbeitsdateien sollten den gleichen Umfang haben oder sich im Umfang nur wenig unterscheiden.

24.6. Steuerkarten für das Sortieren

Jede zum Sortieren benötigte Steuerkarte weist neben mehreren Operanden ein Operationsfeld auf. Vor und nach diesem müssen ein oder mehrere Leerstellen verbleiben. Die Operanden sind voneinander durch Kommas zu trennen; sie werden also wie die Operanden in der AL niedergeschrieben. Im Bedarfsfall können die Statements fortgesetzt werden; hierzu ist irgendeine Lochung auf Spalte 72 zu praktizieren und die nächste Karte mit Spalte 16 zu beginnen.

Die Ordnung, nach der der Datenbestand zu sortieren ist, ist durch eines der beiden folgenden Statements zu umreißen:

SORT FIELDS=(p_1,m_1,f_1,s_1,···,p_n,m_n,f_n,s_n) [,SIZE=y]
SORT FIELDS=(p_1,m_1,s_1,···,p_n,m_n,s_n),FORMAT=f [,SIZE=y]

Durch jede Wertequadrupel p,m,f,s bzw. Wertetripel p,m,s ist ein Feld beschrieben, nach dem das Sortieren durchzuführen ist. Als oberster Sortierbegriff gilt das erste Feld p_1, dann folgt das zweite p_2 usw. Maximal können 64 Felder genannt werden, nach denen sortiert werden soll. Die Summe aller Feldlängen darf dabei 256 Bytes nicht überschreiten.

Durch jedes p_i wird der Feldbeginn angezeigt, p_i ist relativ zum Byte 1 des Satzes anzugeben. Handelt es sich um ein Binärfeld und beginnt dieses nicht an einer Bytegrenze, so ist die Anfangsbitstelle des Feldes innerhalb des betreffenden Bytes durch eine Dezimalziffer anzugeben. Somit zeigt 2.3 an, daß das Feld auf der 3. Bitstelle des Bytes 2, d.h. auf der 12. Bitstelle des Satzes anfängt.

Jedes m_i verweist auf die Länge des Sortierbegriffes in Bytes. Bei einem Binärfeld ist eine gebrochene Länge in der Dezimalschreibweise auszudrücken. Somit besagt z.B. 1.5, daß es sich um ein Feld handelt, das aus 13 Bitstellen besteht. Die Nachpunktziffer darf natürlich nicht größer als 7 sein.

Jedes f_i sagt etwas über das Datenformat aus. Folgende Abkürzungen sind gebräuchlich:

CH	Zeichen (sortiert nach der Reihenfolge, die sich aus dem EBCDI-Code ergibt)
BI	Binärzeichen (sortiert als vorzeichenlose Bits)
ZD	dezimal mit Zone (sortiert in numerischer Folge)
PD	dezimal gepackt (sortiert in numerischer Folge)
FI	dual in Festpunktdarstellung (sortiert in numerischer Folge)
FL	Gleitpunktdarstellung (sortiert in numerischer Folge)

Sind die f_i-Angaben aller Sortierfelder gleich, so kann die zweite Form des SORT-Statements benutzt werden. Gebrochene Werte können nur für Felder angegeben werden, die zum BI-Format gehören.

Für s_i kann entweder A (aufsteigende Reihenfolge) oder D (fallende Reihenfolge) geschrieben werden.

Der bei SIZE anzugebende Wert ist gleich der exakten Zahl der Sätze in der zu sortierenden Datenmenge. Ist diese nicht bekannt, d.h. muß sie geschätzt werden, so muß dem Schätzwert der Buchstabe E vorausgehen. Somit ist durch

SIZE=E3ØØØ

gesagt, daß der zu sortierende Datenbestand schätzungsweise aus 3000 Sätzen besteht. Jede Schätzung sollte höher liegen als die tatsächliche Satzzahl. Fehlt die Angabe von SIZE, so nimmt das Sortierprogramm eine der folgenden Aussagen an:

a) Der gesamte Datenbestand paßt auf eine Magnetbandrolle, gleichgültig, welche Sortierblöcke dabei benutzt werden,
b) der Arbeitsplatz auf Plattenspeichern ist dem zu sortierenden Datenbestand angemessen.

Bei einigen Sortierprogrammen heißt das Schlüsselwort dieses Operanden FILSZ und nicht SIZE.

Doch sehen wir uns zunächst einmal ein typisches Sortierstatement an:

```
SORT   FIELDS=(3,2,PD,A,1,2,PD,A,5,2,CH,A),SIZE=E5ØØØ
```

Die meisten Sortierprogramme im Betriebssystem DOS erfordern die Angabe des Operanden

```
WORK=n
```

Unter n ist die Zahl der vorgesehenen Arbeitsdateien zu nennen. Bei der Benutzung von Bändern zur Speicherung der Arbeitsdateien muß n größer als oder gleich 3 sein; bei Benutzung von Plattenspeichern kann n irgendeinen Wert zwischen 1 und 9 inklusive annehmen.

Alle Sortierprogramme im DOS und einige im OS erfordern das RECORD-Statement in einer der beiden folgenden Formen:

```
RECORD   TYPE=F,LENGTH=(l1,l2,l3)
RECORD   TYPE=V,LENGTH=(l1,l2,l3,l4,l5)
```

Sätze, die im U-Format vorliegen, können nicht sortiert werden. Beim RECORD-Statement bedeuten:

l_1	Maximallänge des Eingabesatzes (bei einigen OS-Sortierprogrammen tritt an Stelle von l_1 die Standardlänge, die aus dem LRECL-Operanden des DD-Statements genommen wird, bei den DOS-Sortierprogrammen kann dieser Operand nicht weggelassen werden)
l_2	Länge der Eingabesätze, die zum Ausgang E15 gehören; fehlt l_2, so wird $l_2=l_1$ angenommen
l_3	Länge der Ausgabesätze bei E35; fehlt l_3, so wird $l_3=l_2$ angenommen
l_4	Minimallänge der Eingabesätze; fehlt l_4, so wird angenommen, daß dieser Wert gleich der Nummer der Stelle ist, die das rechtsbündige Byte des rechtsbündigen Steuerfeldes belegt
l_5	Häufigste Eingabesatzlänge; fehlt diese Angabe, so wird der Mittelwert von l_1 und l_4 benutzt

Die Steuerkarte

```
END
```

ohne Operanden kann benutzt werden, um das Ende des Steuerkartenpaketes anzuzeigen.

Die folgenden drei Steuerkarten gelten nur für die im Betriebssystem DOS verfügbaren Sortierprogramme. Im Betriebssystem OS liegen hierfür die Informationen in den EXEC-. und DD-Statements vor.

Die erste dieser drei Steuerkarten kann in zwei Fassungen gebracht werden:

```
INPFIL   [BLKSIZE=n][,VOLUME=m][,OPEN=option][,CLOSE=option]
INPFIL   EXIT
```

Die zweite Form ist dann zu verwenden, wenn die zu sortierenden Sätze nur über den Ausgang E15 einlaufen. Zur ersten Form sind folgende Aussagen zu machen:

a) Blocklänge
Fehlt BLKSIZE, so wird BLKSIZE=l_1 für die Satzart F und BLKSIZE=l_1+4 für die Satzart V angenommen.

b) Datenträgerzahl
Durch VOLUME wird die Zahl der Magnetbandrollen angegeben, auf denen der Eingabedatenbestand residiert; bei Plattenspeichern wird dieser Operand ignoriert. Bei

Die Ordnung, nach der der Datenbestand zu sortieren ist, ist durch eines der beiden folgenden Statements zu umreißen:

SORT FIELDS=(p_1,m_1,f_1,s_1,···,p_n,m_n,f_n,s_n) [,SIZE=y]
SORT FIELDS=(p_1,m_1,s_1,···,p_n,m_n,s_n),FORMAT=f [,SIZE=y]

Durch jede Wertequadrupel p,m,f,s bzw. Wertetripel p,m,s ist ein Feld beschrieben, nach dem das Sortieren durchzuführen ist. Als oberster Sortierbegriff gilt das erste Feld p_1, dann folgt das zweite p_2 usw. Maximal können 64 Felder genannt werden, nach denen sortiert werden soll. Die Summe aller Feldlängen darf dabei 256 Bytes nicht überschreiten.

Durch jedes p_i wird der Feldbeginn angezeigt, p_i ist relativ zum Byte 1 des Satzes anzugeben. Handelt es sich um ein Binärfeld und beginnt dieses nicht an einer Bytegrenze, so ist die Anfangsbitstelle des Feldes innerhalb des betreffenden Bytes durch eine Dezimalziffer anzugeben. Somit zeigt 2.3 an, daß das Feld auf der 3. Bitstelle des Bytes 2, d.h. auf der 12. Bitstelle des Satzes anfängt.

Jedes m_i verweist auf die Länge des Sortierbegriffes in Bytes. Bei einem Binärfeld ist eine gebrochene Länge in der Dezimalschreibweise auszudrücken. Somit besagt z.B. 1.5, daß es sich um ein Feld handelt, das aus 13 Bitstellen besteht. Die Nachpunktziffer darf natürlich nicht größer als 7 sein.

Jedes f_i sagt etwas über das Datenformat aus. Folgende Abkürzungen sind gebräuchlich:

CH	Zeichen (sortiert nach der Reihenfolge, die sich aus dem EBCDI-Code ergibt)
BI	Binärzeichen (sortiert als vorzeichenlose Bits)
ZD	dezimal mit Zone (sortiert in numerischer Folge)
PD	dezimal gepackt (sortiert in numerischer Folge)
FI	dual in Festpunktdarstellung (sortiert in numerischer Folge)
FL	Gleitpunktdarstellung (sortiert in numerischer Folge)

Sind die f_i-Angaben aller Sortierfelder gleich, so kann die zweite Form des SORT-Statements benutzt werden. Gebrochene Werte können nur für Felder angegeben werden, die zum BI-Format gehören.

Für s_i kann entweder A (aufsteigende Reihenfolge) oder D (fallende Reihenfolge) geschrieben werden.

Der bei SIZE anzugebende Wert ist gleich der exakten Zahl der Sätze in der zu sortierenden Datenmenge. Ist diese nicht bekannt, d.h. muß sie geschätzt werden, so muß dem Schätzwert der Buchstabe E vorausgehen. Somit ist durch

SIZE=E3ØØØ

gesagt, daß der zu sortierende Datenbestand schätzungsweise aus 3000 Sätzen besteht. Jede Schätzung sollte höher liegen als die tatsächliche Satzzahl. Fehlt die Angabe von SIZE, so nimmt das Sortierprogramm eine der folgenden Aussagen an:

a) Der gesamte Datenbestand paßt auf eine Magnetbandrolle, gleichgültig, welche Sortierblöcke dabei benutzt werden,
b) der Arbeitsplatz auf Plattenspeichern ist dem zu sortierenden Datenbestand angemessen.

Bei einigen Sortierprogrammen heißt das Schlüsselwort dieses Operanden FILSZ und nicht SIZE.

Doch sehen wir uns zunächst einmal ein typisches Sortierstatement an:

```
SORT   FIELDS=(3,2,PD,A,1,2,PD,A,5,2,CH,A),SIZE=E5000
```

Die meisten Sortierprogramme im Betriebssystem DOS erfordern die Angabe des Operanden

```
WORK=n
```

Unter n ist die Zahl der vorgesehenen Arbeitsdateien zu nennen. Bei der Benutzung von Bändern zur Speicherung der Arbeitsdateien muß n größer als oder gleich 3 sein; bei Benutzung von Plattenspeichern kann n irgendeinen Wert zwischen 1 und 9 inklusive annehmen.

Alle Sortierprogramme im DOS und einige im OS erfordern das RECORD-Statement in einer der beiden folgenden Formen:

```
RECORD   TYPE=F,LENGTH=(l1,l2,l3)
RECORD   TYPE=V,LENGTH=(l1,l2,l3,l4,l5)
```

Sätze, die im U-Format vorliegen, können nicht sortiert werden. Beim RECORD-Statement bedeuten:

l_1	Maximallänge des Eingabesatzes (bei einigen OS-Sortierprogrammen tritt an Stelle von l_1 die Standardlänge, die aus dem LRECL-Operanden des DD-Statements genommen wird, bei den DOS-Sortierprogrammen kann dieser Operand nicht weggelassen werden)
l_2	Länge der Eingabesätze, die zum Ausgang E15 gehören; fehlt l_2, so wird $l_2=l_1$ angenommen
l_3	Länge der Ausgabesätze bei E35; fehlt l_3, so wird $l_3=l_2$ angenommen
l_4	Minimallänge der Eingabesätze; fehlt l_4, so wird angenommen, daß dieser Wert gleich der Nummer der Stelle ist, die das rechtsbündige Byte des rechtsbündigen Steuerfeldes belegt
l_5	Häufigste Eingabesatzlänge; fehlt diese Angabe, so wird der Mittelwert von l_1 und l_4 benutzt

Die Steuerkarte

```
END
```

ohne Operanden kann benutzt werden, um das Ende des Steuerkartenpaketes anzuzeigen.

Die folgenden drei Steuerkarten gelten nur für die im Betriebssystem DOS verfügbaren Sortierprogramme. Im Betriebssystem OS liegen hierfür die Informationen in den EXEC-, und DD-Statements vor.

Die erste dieser drei Steuerkarten kann in zwei Fassungen gebracht werden:

```
INPFIL   [BLKSIZE=n][,VOLUME=m][,OPEN=option][,CLOSE=option]
INPFIL   EXIT
```

Die zweite Form ist dann zu verwenden, wenn die zu sortierenden Sätze nur über den Ausgang E15 einlaufen. Zur ersten Form sind folgende Aussagen zu machen:

a) Blocklänge
Fehlt BLKSIZE, so wird BLKSIZE=l_1 für die Satzart F und BLKSIZE=l_1+4 für die Satzart V angenommen.

b) Datenträgerzahl
Durch VOLUME wird die Zahl der Magnetbandrollen angegeben, auf denen der Eingabedatenbestand residiert; bei Plattenspeichern wird dieser Operand ignoriert

Magnetbändern ohne Kennsätze muß die genaue Anzahl der Rollen genannt sein; bei Magnetbändern mit Standardkennsätzen kann irgendeine Zahl unter m aufgeführt werden, die mindestens gleich der Anzahl der Rollen ist. Fehlt der Operand VOLUME, so wird VOLUME=1 angenommen.

c) Eröffnen
Als Standard gilt

```
OPEN=RWD
```

wenn dieser Operand fehlt, d.h. Zurückspulen der Rollen zur Eröffnungszeit. Die einzige andere mögliche Angabe lautet NORWD (kein Zurückspulen zur Eröffnungszeit).

d) Schließen
Als Standard gilt

```
CLOSE=RWD,
```

wenn dieser Operand weggelassen ist. Daneben können

```
CLOSE=NORWD
```

und

```
CLOSE=UNLD,
```

d.h. Zurückspulen und Entladen angegeben werden. Das IPFIL-Statement kann entfallen, wenn alle seine Operanden weggelassen werden können.

Das zweite Statement kann ebenfalls in zwei Fassungen gebraucht werden:

```
OUTFILE  [BLKSIZE=m][,NOTPMK][,OPEN=option][,CLOSE=option]
OUTFILE  EXIT
```

Die zweite Form ist dann zu benutzen, wenn alle Sätze über den Ausgang-E35 zur Verfügung gestellt werden. Im übrigen werden durch OUTFIL die gleichen Informationen über die Ausgabedatenmenge zur Verfügung gestellt wie es durch INPFIL für den Eingabedatenbestand geschieht. Die Standards sind ebenfalls die gleichen wie beim INPFIL-Statement. Der Operand NOTPMK verhindert das Schreiben einer führenden Bandmarke auf Bänder ohne Kennsätze. Das OUTFIL-Statement kann bei Fehlen aller Operanden einfach weggelassen werden.

Das dritte Statement lautet

```
OPTION  [PRINT=option][,STORAGE=n][,LABEL=(ausgabe,eingabe,arbeit)]
```

Die drei Operanden besagen folgendes:

a) Drucken

PRINT=NONE	Das Sortierprogramm druckt keine Nachrichten
PRINT=CRITICAL	Nur diejenigen Nachrichten werden gedruckt, die kritisch in bezug auf den Ablauf des Sortierprogrammes sind
PRINT=ALL	Es werden sowohl Fehlernachrichten als auch informative Nachrichten gedruckt

Standard bei Weglassung dieses Operanden ist ALL.

b) Speicher
Durch den Operanden STORAGE wird die Anzahl der Bytes des virtuellen Speichers festgelegt, die von dem Sortierprogramm beansprucht werden können. Fehlt dieser

Operand, so wird die gesamte Partition benutzt. Wird für n ein Wert angegeben, der größer als die Zahl der verfügbaren Bytes ist, so wird nur der verfügbare Platz benutzt.

c) Kennsätze
Für eingabe, ausgabe, arbeit (Eingabedateien, Ausgabedateien, Arbeitsdateien) des Operanden LABEL können folgende Angaben gemacht werden:

N	nichtstandardisierte Kennsätze
S	standardisierte Kennsätze
U	keine Kennsätze

Fehlt dieser Operand, so wird

```
LABEL=(S,S,S)
```

angenommen.

Diese Betrachtungen sollen durch ein Beispiel abgeschlossen werden. Es sind die folgenden vier Sätze (Inhalt in Sedezimalziffern) zu sortieren:

```
345C 89ØD C3C4 F3FØ
345F 891D D3C5 F3F1
345C ØØ2C E2F4 F3F2
345C 891D D4C8 F3F3
```

Als Steuerkarten können folgende vorliegen:

```
SORT     FIELDS=(3,2,PD,A,1,2,PD,A,5,2,CH,A,),SIZE=4,WORK=1
RECORD   TYPE=F,LENGTH=(8)
INPFIL   VOLUME=1,OPEN=RWD,CLOSE=RWD,BLKSIZE=8
OUTFIL   BLKSIZE=8,OPEN=RWD,CLOSE=UNLD
OPTION   PRINT=CRITICAL,STORAGE=3ØØØØ
END
```

Durch den so aufgesetzten Sortierlauf werden die vier Sätze in folgender Reihenfolge zurückgegeben:

```
345F 891D D3C5 F3F1
345C 891D D4C8 F3F3
345C 89ØD C3C4 F3FØ
345C ØØ2C E2F4 F3F2
```

Zusätzlich erledigen die meisten Sortierprogramme noch folgende Arbeiten:

a) Richtigstellung der Vorzeichen bei Dezimalzahlen, die gepackt oder mit Zonen versehen sind,
b) Normalisierung aller Zahlen in Gleitpunktdarstellung.

Voraussetzung hierfür ist natürlich, daß diese Zahlen zu Sortierfeldern gehören.

24.7. Sortieren unter dem Betriebssystem DOS (Statements der Jobsteuersprache)

Die Dateinamen und die SYS-Einheiten, die vom Sortierprogramm benutzt werden, sind in Abb. 24.7. aufgeführt.

Dateiattribut	*Dateiname*	*Keine Ausgänge*	*Ausgänge*		
			E15	E35	E15 u. E35
OUTPUT	SORTOUT	SYSØØ1	SYSØØ1	—	—
INPUT	SORTIN1	SYSØØ2	—	SYSØØ2	—
WORK	SORTWK1 bis SORTWK9	SYSØØ3 bis SYSØ11	SYSØØ2 bis SYSØ1Ø	SYSØØ2 bis SYSØ1Ø	SYSØØ1 bis SYSØØ9

Abb. 24.7. Die logischen Systemeinheiten und die vom Programmierer vergebenen Einheiten für das Sortieren unter dem Betriebssystem DOS

Das Sortierprogramm wird unter dem Namen SORT aufgerufen. Die Eingabe- und Ausgabedateien können auf Magnetbändern und/oder Plattenspeichern residieren; jede Kombination ist möglich. Es ist jedoch nicht erlaubt, daß das gleiche Band bzw. der gleiche Plattenspeicherbereich für Eingabe-, Ausgabe- und Arbeitsdateien benutzt wird. Folgende Kombinationen sind jedoch gestattet:

(1) SORTOUT und SORTIN1
(2) SORTOUT und SORTWK1
(3) $SORTIN_1$ und SORTWKm
(m verweist auf die letzte Arbeitsdatei)

Die letztere Möglichkeit besteht nur bei Bändern. Wird ein Plattenspeicherbereich für zwei Arbeitsdateien benutzt, so müssen beide haargenau den gleichen Bereich besetzen und dieselbe Dateikennzeichnung aufweisen. Eine typische Folge von Jobsteuerstatements lautet:

```
// JOB SORTI
// ASSGN  (für SYSØØ1 bis SYSØØ5)
// TLBL   SORTIN ···
// TLBL   SORTOUT ···
// EXEC SORT
   Steuerkarten für WORK=3 und LABEL=(S,S,U)
/*
/&
```

24.8. Sortieren unter dem Betriebssystem OS (Statements der Jobsteuersprache)

Das Sortierprogramm wird durch das Statement

```
//name   EXEC   PGM=SORT,PARM=···
```

aufgerufen. Als Parameter können

```
       CORE=xxxxxx
und    MSG=option
```

benutzt werden. Durch CORE wird der Umfang des Speicherplatzes festgelegt, den das Sortierprogramm benutzen kann. Durch MSG wird die Art der Druckausgabe angezeigt. Folgende Angaben sind möglich.

MSG=NO	Es werden keine Nachrichten gedruckt
MSG=CC	Kritische Nachrichten werden über die Konsole ausgegeben
MSG=CP	Kritische Nachrichten werden über den Drucker ausgegeben
MSG=AC	Alle Nachrichten werden über die Konsole ausgegeben
MSG=AP	Alle Nachrichten werden über den Drucker ausgegeben

Standardangaben sind vom Systemprogrammierer zu definieren. Das nachfolgende Statement ist ein Beispiel für ein EXEC-Statement:

```
//ST   EXEC   PGM=SORT,PARM='CORE=6ØØØØ,MSG=AP'
```

Folgende DD-Statements werden benötigt:

SORTOUT	Ausgabe nach dem Sortieren; dieses Statement entfällt, wenn über den Ausgang E35 die Ausgabe erfolgt
SORTIN	Eingabe für das Sortierprogramm; dieses Statement entfällt, wenn die Eingabe über E15 erfolgt
SORTWKØ1 bis SORTWK32	Arbeitsdateien; alle Arbeitsdateien müssen entweder Bändern zugeordnet sein, oder sie müssen zu Plattenspeicherbereichen gehören. Keine Überlappung zwischen SORTIN, SORTOUT und den Arbeitsdateien möglich
SYSOUT	Druckerausgabe vom Sortierprogramm
SORTLIB	Datenmenge, die die Lademoduln des Sortierprogrammes enthält (normalerweise SYS1.SORTLIB).
SYSIN	Datei, über die die Steuerkarten für das Sortierprogramm einlaufen

Die DD-Statements für SORTIN und SORTOUT müssen die DCB-Unterparameter RECFM, BLKSIZE und LRECL enthalten; DEN kann ebenfalls spezifiziert werden. Bei einigen Sortierprogrammen können diese Angaben fehlen, wenn der Katalog oder die Kennsätze die erforderlichen Informationen enthalten. UNIT und SPACE müssen für Arbeitsdateien angegeben werden, wenn diese Plattenspeichern zugeordnet sind, UNIT für Arbeitsdateien auf Bändern. Mindestens drei Arbeitsdateien sind zu spezifizieren. Für SORTOUT ist außerdem die Angabe des Parameters DISP erforderlich.

Nur die Steuerkarte SORT ist erforderlich; END kann zusätzlich aufgeführt werden. Die Steuerkarte RECORD ist nur dann notwendig, wenn die Standardwerte für l_2 bis l_5 nicht akzeptabel sind (siehe Abschnitt 24.6.). Ein typisches Job-Steuerkartenpaket ist in Abb. 24.8. aufgelistet.

```
//S EXEC PGM=SORT,PARM='CORE=60000,MSG=AP'
//SYSOUT    DD  SYSOUT=A
//SORTLIB   DD  DISP=SHR,DSNAME=SYS1.SORTLIB
//SORTIN    DD  DISP=OLD,DCB=(RECFM=FB,BLKSIZE=800,LRECL=80),
                UNIT=2400,VOL=SER=123321,DSNAME=INPUT.A
//SORTOUT   DD  DISP=(NEW,KEEP),DCB=(REFM=FB,BLKSIZE=800,LRECL=80),
                UNIT=2400,VOL=SER=234432,DSNAME=OUTPUT.A
//SORTWK01  DD  UNIT=2400
//SORTWK02  DD  UNIT=2400
//SORTWK03  DD  UNIT=2400
//SYSIN     DD  *
   SORT  FIELDS=(1,6,A),FORMAT=CH,SIZE=E40000
/*
```

Abb. 24.8. Beispiel für ein Steuerkartenpaket zur Durchführung des Sortierens

24.9. Sortieren in PL/1-Programmen

Ein PL/1-Programm kann das Sortierprogramm genau so aufrufen wie jedes andere Unterprogramm. Die hierzu erforderliche CALL-Anweisung kann in folgenden vier Varianten auftreten:

CALL PLISRTA (a_1,a_2,a_3,a_4,a_7,a_8) ;

CALL PLISRTB (a_1,a_2,a_3,a_4,a_5,a_7,a_8) ;

CALL PLISRTC (a_1,a_2,a_3,a_4,a_6,a_7,a_8) ;

CALL PLISRTD (a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8) ;

Die beim Aufruf mitzugebenden Argumente a_1 bis a_8 besitzen folgende Bedeutung:

a_1	Zeichenkette mit dem SORT-Steuerstatement
a_2	Zeichenkette mit dem RECORD-Steuerstatement
a_3	a) DOS: Zeichenkette mit dem OPTION-Steuerstatement b) OS: Umfang des vom Sortierprogramm zu benutzenden Hauptspeicherplatzes in Form eines numerischen Wertes (CORE=wert)
a_4	Variable mit den Attributen BINARY FIXED(31) zur Aufnahme des Rückkehrcodes[1)]
a_5	Eingangsname der Prozedur für den Ausgang E15
a_6	Eingangsname der Prozedur für den Ausgang E35
a_7	a) DOS: Zeichenkette mit dem INPFIL-Steuerstatement b) OS: Eine Zeichenkette der Länge 4, die die Buchstabenfolge enthält, die an Stelle der Buchstabenfolge SORT in den DD-Statements für SORTOUT, SORTIN und SORTWKXX benutzt wird. Fehlt dieses Argument, so gilt SORT als benutzt
a_8	a) DOS: Zeichenkette mit dem OUTFIL-Steuerstatement b) OS: Eine Zeichenkette der Länge 2, die den Wert des MSG-Parameters des EXEC-Statements enthält. Fehlt dieses Argument, so wird der Standardwert der Installation benutzt

1) Dieser Variablen weist das Sortierprogramm den Wert 0 zu, wenn das Sortieren erfolgreich ablief, anderenfalls den Wert 16, wenn der Sortierversuch erfolglos blieb.

Wird PLISRTA aufgerufen, so liest das Sortierprogramm die Eingabedatenmenge, sortiert die Sätze und zeichnet die Ausgabedatenmenge auf. Wird PLISRTB aufgerufen, so liest das Sortierprogramm die Sätze durch Unterstützung der Routine, die bei E15 angeschlossen ist; die Standardeingabedatenmenge wird nicht benutzt. Beim Aufruf der Eingangsstelle PLISRTC liest das Sortierprogramm zwar die Sätze des Eingabedatenbestandes, die Ausgabedatensätze werden jedoch über den Ausgang E35 präsentiert; eine Ausgabedatenmenge wird also nicht geschrieben. Beim Aufruf von PLISRTD erfolgt die Eingabe über E15, die Ausgabe über E35. Bei allen vier Fällen wird a_4 am Ende des Sortierlaufes vom Sortierprogramm der Wert 0 oder der Wert 16 zugewiesen.

Bei Benutzung der Eingangsstelle PLISRTA sind keine besonderen Vorkehrungen zu treffen. Hingewiesen sei jedoch auf die Tatsache, daß der für das Sortierprogramm verfügbare Speicherplatz um den Betrag herabgesetzt wird, den das während des Sortierens im Speicher verbleibende PL/1-Programm selbst beansprucht.

Wird die Eingabe über E15 bewerkstelligt, so erhält die unter a_5 genannte Benutzerroutine immer dann die Ablaufsteuerung, wenn das Sortierprogramm bereit ist, einen weiteren Satz zu verarbeiten. Die Benutzerroutine wird gewöhnlich als separate Prozedur geschrieben, die entweder intern oder extern zu der Prozedur ist, die das Sortierprogramm aufruft. Diese Routine muß auf das Statement

```
CALL PLIRETC(n) ;
```

stoßen, über das dem Sortierprogramm ein Rückkehrcode gemeldet werden muß. Bei n=12 gibt die Benutzerroutine an das Sortierprogramm einen Satz zurück. Nach dessen Verarbeitung bekommt die Benutzerroutine erneut die Steuerung. Durch n=8 wird dem Sortierprogramm signalisiert, daß das Ende der Eingabe erreicht ist und kein Satz überstellt wird. Die Benutzerroutine für E15 ist als Funktion abzufassen, die als Funktionswert eine Zeichenkette an das Sortierprogramm übergibt. Auf diese kann beim Rückkehrcode n=8 verzichtet werden. Wenn die an E15 angehängte Benutzerroutine Zeichenketten variabler Länge abliefern soll, so muß die Prozedur das Attribut RETURNS(n) enthalten. Hierbei bedeutet n die maximale Satzlänge. Wir präsentieren hierzu ein Beispiel (Abb. 24.9.).

```
RETURNS ( CHARACTE(n) VARYING )
E15V:  PROCEDURE RETURNS (CHARACTER(8Ø) VARYING) ;
         DECLARE  S CHARACTER(8Ø) VARYING ;
         .
         .
         .
         CALL PLIRETC (12) ;
         RETURN(S) ;
         .
         .
         .
         CALL PLIRETC(8) ;
         RETURN ;
         .
         .
         .
       END E15V ;
```

Abb. 24.9. Übersicht über eine an E15 anzuhängende Benutzerroutine

Werden vom Sortierprogramm Sätze an eine Benutzerroutine über den Ausgang E35 übergeben, so muß diese Routine durch Aufruf von PLIRETC ebenfalls einen Rückkehrcode setzen. Ein Rückkehrcode von 4 besagt, daß der nächste Satz über den Ausgang E35 bereitgestellt werden soll. Dagegen betont der Wert 8, daß keine weiteren Sätze mehr über den Ausgang E35 erwartet werden. Die an E35 anzuhängende Benutzerroutine wird normalerweise als Prozedur geschrieben. Sie erhält als Argument vom Sortierprogramm eine Zeichenkette. Für Sätze fester Länge resultiert daraus eine einfache Codieranleitung, nämlich die folgende:

```
E35F:  PROCEDURE (S) ;
         DECLARE  S CHARACTER(8Ø) ;
           .
           .
           .
```

Liegen Sätze variabler Länge vor, so kann die Benutzerroutine selbstverständlich nur eine einstellbare Zeichenkette empfangen. Das nachfolgende Beispiel zeigt dies deutlich:

```
E35V:  PROCEDURE (S) ;
         DECLARE  S CHARACTER(*) ;
           .
           .
           .
```

Die Länge des Parameters 5 wird damit vom Sortierprogramm als Teil des Arguments übergeben.

Nach Durchführung des Sortierens wird die Programmablaufsteuerung stets an das Statement übergeben, das auf die Anweisung

```
CALL PLISRTX ;
```

folgt. Dieses Statement sollte den Rückkehrcode überprüfen. Gewöhnlich führen die folgenden Fakten zu Fehlern beim Ablauf der Sortierprozedur:

a) Widersprüchliche Steuerstatements für das Sortierprogramm,
b) Fehlende oder falsche Steuerstatements der Jobsteuersprache,
c) Ungeeigneter (unzureichender) Arbeitsplatz auf Plattenspeichern.

Sind die Benutzerroutinen, die an die Sortierausgänge angehängt werden, externe Prozeduren in bezug auf die das Sortierprogramm selbst aufrufende Prozedur, so müssen die Ausgangsprozeduren zusammen mit der Hauptprozedur gebunden werden. Da der aus dem PL/1-Programm generierte Maschinencode den für das Sortierprogramm verfügbaren Speicher reduziert, wächst die Sortierzeit an.

24.10. Durch COBOL-Programme veranlaßtes Sortieren

Das Sortierprogramm kann ebenfalls durch Programme aufgerufen werden, die in COBOL geschrieben sind. Zum Aufruf des Sortierprogrammes ist das folgende Statement zu codieren:

```
SORT dateiname-1  {ON  [DESCENDING]  KEY  {datenname}  ...  } ...
                  [ASCENDING]
                  {INPUT  PROCEDURE IS kapitel-1  [THRU kapitel-2]}
                  {USING  dateiname-2}
                  {OUTPUT PROCEDURE IS kapitel-3  [THRU kapitel-4]}
                  {GIVING dateiname-3}
```

Die Gesamtzahl der in ON-Klauseln aufgeführten Datennamen darf 12 nicht übersteigen. In der Reihenfolge müssen sie so angeordnet sein, daß der höchste Sortierbegriff vorn steht, die übrigen folgen gemäß ihrer Rangfolge hintereinander. Für jeden so benutzten Datennamen muß eine Eintragung in einer Satzbeschreibung der Datei mit dem Namen dateiname-1 vorliegen. Außerdem muß gewährleistet sein, daß jeder Datenname ein Element kennzeichnet, das innerhalb der ersten 4092 Stellen des Satzes liegt. Aus den in den ON-Klauseln stehenden Informationen wird das SORT-Statement für das Sortierprogramm generiert.

Die zu numerischen Datennamen gehörenden Werte werden algebraisch sortiert, die zu nichtnumerischen Datennamen gehörenden in der Reihenfolge des EBCDI-Codes.

Durch die USING-Klausel ist die zu sortierende Eingabedatei bestimmt. Diese wird durch das Sortierprogramm selbst eröffnet, verarbeitet und auch abgeschlossen. Die GIVING-Klausel hingegen hält die Ausgabedatei fest, die ebenfalls vom Sortierprogramm eröffnet, geschrieben und schließlich auch abgeschlossen wird. Keine dieser beiden Dateien darf also im Augenblick der Ausführung des SORT-Statements eröffnet sein.

Nach Durchführung des Sortierens wird das Programm mit dem nächsten Statement fortgeführt.

In der Klausel INPUT PROCEDURE ist der Name der Routine zu nennen, die die Eingabesätze für das Sortieren liefern soll, wenn auf ihre Zuführung aus einer Datei verzichtet wird oder werden muß (Ausgang E15). Im Prinzip das gleiche ist für den Ausgang E35 zu sagen. An Stelle des Hineinstellens der sortierten Sätze in eine Ausgabedatei werden diese an eine Routine übergeben, deren Name in der Klausel OUTPUT PROCEDURE anzugeben ist. In beiden Fällen werden die zur Routine gehörenden, im Statement genannten Kapitel (kapitel-1, kapitel-3 usw.) so ausgeführt, als ob sie von einem PERFORM-Statement aufgerufen würden.

Die Eintragung dateiname-1 beinhaltet die Nennung der Ansammlung der für das Sortieren benutzten Arbeitsdateien.

Als Randbemerkung wollen wir an dieser Stelle die folgende einschieben:

Das Statement SORT ist im Subset-Kompilierer des Betriebssystemes DOS nicht verfügbar.

Die Arbeitsdateien sind für den Maschinenteil wie folgt zu deklarieren:

```
SELECT dateiname-1
       ASSIGN TO [gz-1] {systemnam-1} ...
       [OR systemname-2]
       [FOR MULTIPLE {REEL}]
                     {UNIT}
       [RESERVE {gz-2} ALTERNATE [AREA ].
                {NO }            [AREAS]
```

Wird das Wort OR benutzt, so wird es ignoriert. Unter dateiname-1 ist der gleiche Name aufzuführen, der auch im SORT-Statement erscheint.

Im Betriebssystem DOS ist unter gz-1 die Anzahl der Arbeitsdateien aufzuführen; standardmäßig sind das eine bei Benutzung von Plattenspeichern, 3 bei Benutzung von Bändern. Im Betriebssystem OS wird die gesamte ASSIGN-Klausel vernachlässigt. Die Systemnamen besitzen die folgende Form:

SYSØØn-gerät-typ-S-SORTWKn

Hierbei bedeutet n die laufende Nummer 1, 2, ··· der Arbeitsdatei.

Die Bedeutung der Klauseln MULTIPLE und REEL ist die gleiche wie bei allen anderen Dateien.

Im Paragraphen I-O CONTROL kann die Klausel

SAME { RECORD / SORT } AREA FOR { dateiname } ···

niedergeschrieben werden. Wird SORT notiert, so wird dieses Wort ignoriert. Bei Nennung von RECORD kann zu einer bestimmten Zeit nur eine der genannten Dateien einen Satz im Satzgebiet haben.

Im Datenteil benutzen wir an Stelle einer FD-Beschreibung eine Sortierdateienbeschreibung, die durch SD angekündigt wird. Sie lautet allgemein:

```
SD  dateiname-1
    [RECORDING MODE IS modus]
    [DATA { RECORD IS / RECORDS ARE } { datenname } ···]
    [RECORD CONTAINS [gz-1 TO] gz-2 CHARACTERS]
    [LABEL { RECORD IS / RECORDS ARE } { STANDARD / OMITTED }].
```

Der Aufzeichnungsmodus (RECORDING MODE) kann nur zu F oder V bestimmt werden. Die gesamte RECORD-Klausel wird als Kommentar betrachtet. Fehlt die Klausel LABEL[1)], so nimmt das Betriebssystem DOS an, daß Standardkennsätze auf Plattenspeichern, dagegen keine Kennsätze bei Magnetbändern vorliegen. Die Satzeintragungen sind die gleichen, die wir schon bei den FD-Beschreibungen kennengelernt haben.

Wird die Klausel INPUT PROCEDURE benutzt, so werden die die Eingabeprozedur bildenden Kaptiel dann angesprungen, wenn das Sortierprogramm bereit ist, den ersten Satz aufzunehmen. Die Eingabeprozedur wird dabei wie jede andere Prozedur betrachtet, die von einer PERFORM-Anweisung angesprochen wird. Eines allerdings muß, obwohl eigentlich selbstverständlich, herausgestellt werden: Die Eingabeprozedur muß selbst frei von jedem SORT-Statement sein. Jeder dem Sortierprogramm zu übergebende Satz muß durch das Statement

RELEASE satzname [FROM bez]

freigegeben werden. Jeder so benannte Satz wird dann eingabemäßig dem Sortierprogramm überstellt und danach sollte die Eingabeprozedur erneut vom Beginn an ablaufen. Die RELEASE-Anweisung ähnelt also der WRITE-Anweisung mit der Ausnahme, daß die

1) Die Kompilierer im Betriebssystem OS, Versionen 2 und 3, erlauben nicht die Verwendung der LABEL- Klausel.

Ausgabe dem Sortierprogramm übergeben wird. Wird durch geeignete Codierung die Programmablaufsteuerung zum Ende der Eingabeprozedur geführt, so schließt das Sortierprogramm daraus, daß keine weiteren Sätze aufzunehmen sind.

Wenn satzname variabel lang ist, so wird die bez zugeordnete Länge als Satzlänge unterstellt, es sei denn, daß die FROM-Klausel fehlt. In diesem Fall wird die Länge vom letzten Statement extrahiert, durch das Daten satzname zugewiesen wurden. Man kann Werte für jede OCCURS-Klausel setzen und satzname auf sich selbst übertragen, um die Satzlänge hinzuzufügen.

Wird die Klausel OUTPUT PROCEDURE benutzt, so wird die in ihr spezifizierte Ausgabeprozedur zu der Zeit ausgeführt, zu der das Sortierprogramm normalerweise die Ausgabe der sortierten Sätze startet. Die Ausgabeprozedur selbst wird nur ein einziges Mal ausgeführt. Jeder Satz wird dem Benutzer zur Verfügung gestellt, wenn er sich des Statements

```
RETURN dateiname-1 RECORD [INTO bez]
       AT END imperativanweisungen.
```

bedient. Jede RETURN-Anweisung fungiert wie eine READ-Anweisung: Sie stellt den nächsten Satz bereit. Unter dateiname-1 ist der gleiche Dateiname zu nennen, der in den SORT- und SD-Statements auftritt. Die Bedingung AT END kommt dann zum Tragen, wenn der letzte Satz zur Verfügung gestellt wird.

Die Programmablaufsteuerung zum regulären Ende der Ausgabeprozedur zu führen, heißt, daß die Ausgabe des Sortierprogrammes abgeschlossen wird und daß die auf die SORT-Anweisung folgende Anweisung ausgeführt wird. Wird eine RETURN-Anweisung außerhalb der Ausgabeprozedur gefunden, so bewirkt sie einen Programmabbruch.

Fünf spezielle Register werden im Zusammenhang mit dem SORT-Statement eingerichtet. Sie heißen:

a) SORT-RETURN
Seine Eigenschaften sind durch

```
PICTURE S9(4) COMPUTATIONAL
```

beschrieben. Dieses Register kann nur als Sendefeld dienen; in einem DISPLAY-Statement ist sein Gebrauch verboten. Dieses Register wird am Ende eines erfolgreichen Ablaufes der Sortierung auf 0 gesetzt, sonst auf 16.

b) SORT-FILE-SIZE
Seine Eigenschaften sind durch

```
PICTURE S9(8) COMPUTATIONAL
```

umrissen. Dieses Register kann nur als Empfangsfeld benutzt werden. Es kann jedoch nicht in Eingabestatements erscheinen. Sein Anfangswert ist gleich Null. Dieses Register enthält zu Beginn des Sortierlaufes die (geschätzte) Anzahl der zu sortierenden Sätze.

c) SORT-CORE-SIZE
Seine Eigenschaften sind durch

```
PICTURE S9(8) COMPUTATIONAL
```

beschrieben. Dieses Register kann nur als Empfangsfeld benutzt werden; in Eingabestatements kann es jedoch nicht auftreten. Sein Anfangswert beträgt Null. Zu

Beginn des Sortierverfahrens enthält es die Anzahl der Speicherplätze, der sich die Sortierprozedur bedienen kann. Wird dieser Wert zu Null vorgegeben, so wird stattdessen der für die Installation kreierte Standardwert benutzt.

d) SORT-MODE-SIZE
Seine Eigenschaften sind durch

```
PICTURE S9(5) COMPUTATIONAL
```

beschrieben. Auch dieses spezielle Register kann nur als Empfangsfeld dienen, von Eingabeanweisungen ist es auszuschließen. Sein Anfangswert ist gleich Null. Dieses Register kommt nur dann zum Tragen, wenn variabellange Sätze (V-Format) sortiert werden sollen. Es enthält die durchschnittliche Satzlänge. Wird Null vorgegeben, so wird der Mittelwert der Längen derjenigen Sätze eingesetzt, die für die Sortierdatei definiert sind.

e) SORT-MESSAGE[1)]
Seine Eigenschaften sind durch

```
PICTURE X(8)
```

beschrieben. Es enthält den Namen der Datei, die die vom Sortierprogramm ausgegebenen Nachrichten aufnimmt. Der zu diesem Register gehörende Anfangswert heißt SYSOUT.

Aufgabengruppe 64

1) Welche Druckausgabe erfolgt durch das nachstehend aufgelistete Programm, wenn die Eingabewerte

a) 5, b) 6, c) 1 und d) 9 lauten?

```
P:  PROCEDURE OPTIONS(MAIN) ;
      DECLARE  (X,Y) DECIMAL FLOAT(16) ;
      ON ENDFILE(SYSIN)  GOTO E ;
    G: GET LIST(X) ;
      Y = D(X) ;
      PUT LIST(X,Y) ;
      GOTO G ;

        D:  PROCEDURE(F) RETURNS(FLOAT(16)) RECURSIVE ;
              DECLARE  F DECIMAL FLOAT(16) ;
              IF  F<3  THEN RETURN(F) ;
                       ELSE RETURN (1+F*D(F-2)) ;
            END D ;

    END P ;
```

Das Attribut RECURSIVE bei der internen Prozedur D zeigt an, daß es sich um eine rekursive Prozedur handelt.

1) Dieses spezielle Register kann nur bei bestimmten Sortierprogrammen angesprochen werden, die unter dem Betriebssystem OS laufen.

2) Schreibe ein Makro DPM mit drei Operanden, das das Produkt der ersten beiden ermittelt und das Ergebnis dem dritten Operanden zuweist. Alle Operanden sollen ganze Dualzahlen doppelter Genauigkeit in Festpunktdarstellung sein. Nur der Teil des Produktes ist zu speichern, der die Hälfte der Stellenzahl belegt, und zwar die rechtsbündige Hälfte. Fehlt ein Operand, so ist dieser gleich Null zu setzen. Es sind die Register 14, 15, 0 und 1 zu benutzen. Im Bedarfsfall sind als weitere Register die Register 2, 3, 4 und 5 heranzuziehen.

3) Setze einen Sortierlauf auf, der die auf Magnetband residierende Datei sortiert, die in Aufgabe 3 der Aufgabengruppe 33 beschrieben ist. Die Sortierbegriffe lauten der Reihe nach Fakultät, Fach und Vorlesung.

4) Das in Abb. 12.3. vorgestellte Problem ist wie folgt zu modifizieren:

   Die Eingabekarten fallen unsortiert an. Es ist eine Eingabeprozedur für das Sortierprogramm zu schreiben, um auf diese Weise ein Magnetband mit Sätzen zu produzieren, bei denen die Rechnungsnummer maßgebend für die Sortierfolge der Sätze ist.

5) Schreibe ein Sortierprogramm, mit dessen Hilfe das in Abb. 17.1. erstellte Magnetband nach Rechnungsnummern sortiert werden kann.

6) Ändere das Programm von Aufgabe 3 der Aufgabengruppe 48 wie folgt ab: Die Eingabekarten können in beliebiger Reihenfolge anfallen. Füge einen Sortierlauf mit Ausgang hinzu, mit dessen Hilfe die Eingabesätze in die richtige Reihenfolge auf ein Magnetband gelangen.

7) Ändere das Problem von Aufgabe 2 der Aufgabengruppe 49 wie folgt ab: Die Druckausgabe soll sortiert nach dem Inhalt der Adreßzeile 3 erfolgen. Benutze dazu einen Sortierausgang. Schreibe das Programm in COBOL!

8) Löse die Aufgabe 7) in PL/1.

Anhang A: Zeichen und Codes

1. Zeichen

1.1. Datenzeichen

Ihre Anzahl entspricht dem Code. Sie werden in

- *Buchstaben*
- *Ziffern*
- *Sonderzeichen*

unterteilt.

Welche Zeichen von diesem Zeichenvorrat gedruckt werden können, hängt vom Gerätetyp und seiner Druckausrüstung ab. Für verschiedene Geräte ist der Vorrat an druckbaren Zeichen in der nachstehenden Abbildung dargestellt.

Zeichen	Locher		Drucker IBM 1443				Andere Zeilendrucker			Sonstige Drucker
	029 129	5496	13	39	52	63	AN A11	HN H11	PN P11	3210 3215
0 bis 9	x	x	x	x	x	x	x	x	x	x
A bis Z, $	x	x		x	x	x	x	x	x	x
a bis z										x
- *	x	x	x		x	x	x	x	x	x
+ / &	x	x			x	x	x	x	x	x
= ' ( )	x	x			x	x		x	x	x
# @ %	x	x			x	x	x		x	x
¬ | " < > : ; _ ?	x	x				x				x
! ¢	x	x				x				x
¤ siehe auch e)					x		x			
(0-8-2) siehe auch d)	x					x				
}		x								

Anmerkungen:

a) Die QN- und QNC-Ausrüstung für die anderen Zeilendrucker enthalten den gleichen Zeichensatz wie die PN- und P11-Ausrüstungen

b) Die AN- und A11-Ausrüstungen drucken folgende Ersatzzeichen:
   - # an Stelle von '
   - @ an Stelle von =
   - % an Stelle von (
   - ¤ an Stelle von)

c) Die HN- und H11-Ausrüstungen drucken folgende Ersatzzeichen:
   - ' an Stelle von #
   - = an Stelle von @
   - (an Stelle von %
   -) an Stelle von ¤

d) Der Code 0-8-2 bedeutet ein Leerzeichen; es wird von den Druckern an dieser Stelle kein Zeichen gedruckt

e) Der Lochkartencode 12-4-8 (80spaltige Lochkarten) bzw. B-A-8-4 (96spaltige Lochkarten) ergibt das Zeichen ¤ (sedezimale Verschlüsselung 4C)

1.2. Gerätesteuerzeichen

Einige Zeichen sind für die Steuerung von Geräten reserviert (Gerätesteuerzeichen). Diese sind teils international, teils herstellerbezogen genormt. Zur Gerätesteuerung werden im allgemeinen Buchstaben und Ziffern nicht herangezogen.

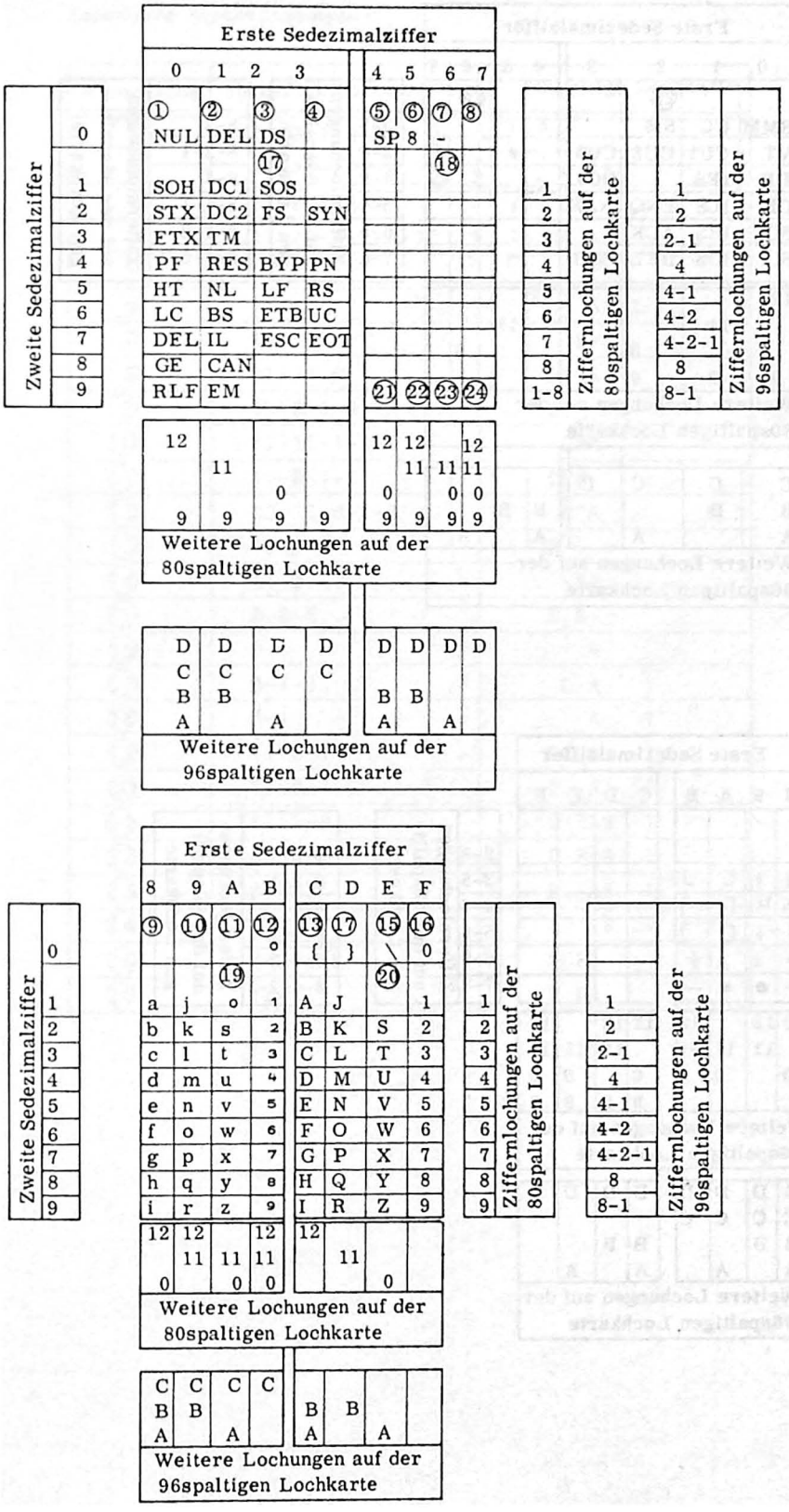
2. Sedezimalcode

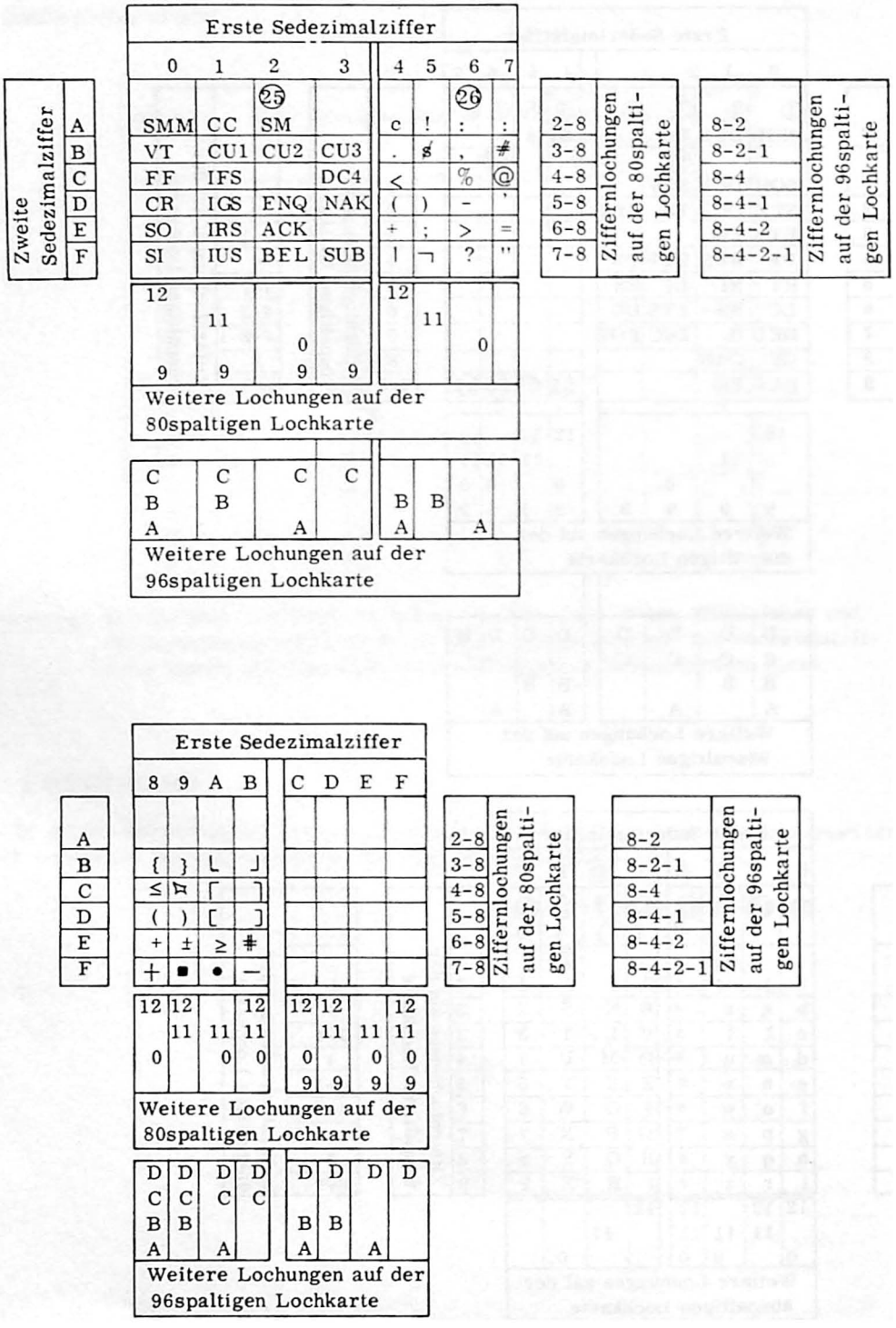
Ziffer	Duale Verschlüsselung	Beim Auftreten in der Vorzeichenstelle wird dieses Zeichen betrachtet als
0	0000	
1	0001	
2	0010	
3	0011	
4	0100	
5	0101	
6	0110	
7	0111	
8	1000	
9	1001	
A	1010	+
B	1011	-
C	1100	+
D	1101	-
E	1110	+
F	1111	+

Anmerkung: Bei der IBM /370 wird die Sedezimalziffer C bei einem Pluszeichen und die Sedezimalziffer D bei einem Minuszeichen erzeugt. Die Sedezimalziffer F kommt als Zonenteil zu vorzeichenfreien Dezimalziffern hinzu.

3. EBCDI-Code

EBCDI ist die Abkürzung von *Extented Binary Coded Decimal Interchange (erweitertes binär codiertes Dezimalsystem für Informationsaustausch).*


Erste Sedezimalziffer
0 1 2 3 4 5 6 7
Zweite Sedezimalziffer
0 NUL DEL DS SP 8 -
1 SOH DC1 SOS
2 STX DC2 FS SYN
3 ETX TM
4 PF RES BYP PN
5 HT NL LF RS
6 LC BS ETB UC
7 DEL IL ESC EOT
8 GE CAN
9 RLF EM
Ziffernlochungen auf der 80spaltigen Lochkarte
Ziffernlochungen auf der 96spaltigen Lochkarte
Weitere Lochungen auf der 80spaltigen Lochkarte
Weitere Lochungen auf der 96spaltigen Lochkarte
Erste Sedezimalziffer
8 9 A B C D E F
Zweite Sedezimalziffer
0 { } \ 0
1 a j o 1 A J 1
2 b k s 2 B K S 2
3 c l t 3 C L T 3
4 d m u 4 D M U 4
5 e n v 5 E N V 5
6 f o w 6 F O W 6
7 g p x 7 G P X 7
8 h q y 8 H Q Y 8
9 i r z 9 I R Z 9
Ziffernlochungen auf der 80spaltigen Lochkarte
Ziffernlochungen auf der 96spaltigen Lochkarte
Weitere Lochungen auf der 80spaltigen Lochkarte
Weitere Lochungen auf der 96spaltigen Lochkarte



Zweite Sedezimalziffer	Erste Sedezimalziffer 0	1	2	3	4	5	6	7
A	SMM	CC	SM		c	!	:	:
B	VT	CU1	CU2	CU3	.	$	,	#
C	FF	IFS		DC4	<		%	@
D	CR	IGS	ENQ	NAK	(	)	-	'
E	SO	IRS	ACK		+	;	>	=
F	SI	IUS	BEL	SUB	|	¬	?	"
	12 9	11 9	0 9	9	12	11	0	
	Weitere Lochungen auf der 80spaltigen Lochkarte							
	C B A	C B	C A	C	B A	B	A	
	Weitere Lochungen auf der 96spaltigen Lochkarte							

2-8	Ziffernlochungen auf der 80spaltigen Lochkarte
3-8	
4-8	
5-8	
6-8	
7-8	

8-2	Ziffernlochungen auf der 96spaltigen Lochkarte
8-2-1	
8-4	
8-4-1	
8-4-2	
8-4-2-1	

	Erste Sedezimalziffer 8	9	A	B	C	D	E	F
A								
B	{	}	└	┘				
C	≤	¤	┌	┐				
D	(	)	[	]				
E	+	±	≥	#				
F	┼	■	•	—				
	12 0	12 11	11 0	12 11 0	12 0 9	12 11 9	11 0 9	12 11 0 9
	Weitere Lochungen auf der 80spaltigen Lochkarte							
	D C B A	D C B	D C A	D C	D B A	D B	D A	D
	Weitere Lochungen auf der 96spaltigen Lochkarte							

2-8	Ziffernlochungen auf der 80spaltigen Lochkarte
3-8	
4-8	
5-8	
6-8	
7-8	

8-2	Ziffernlochungen auf der 96spaltigen Lochkarte
8-2-1	
8-4	
8-4-1	
8-4-2	
8-4-2-1	

Besondere Kartenlochungen

Anmerkung	80spaltige Lochkarte	96spaltige Lochkarte
①	12-Ø-1-8-9	C
②	12-11-1-8-9	C A8 2
③	11-Ø-1-8-9	CB
④	12-11-Ø-1-8-9	DC A
⑤		
⑥	12	A8 2
⑦	11	B
⑧	12-11-Ø	D A
⑨	12-Ø-1-8	DC
⑩	12-11-1-8	CBA
⑪	11-Ø-1-8	DCB
⑫	12-11-Ø-1-8	C A
⑬	12-Ø	D
⑭	11-Ø	BA
⑮	Ø-2-8	D B
⑯	Ø	A
⑰	Ø-1-9	C A 1
⑱	Ø-1	A 1
⑲	11-Ø-1	DC A 1
⑳	11-Ø-1-9	D A 1
㉑	12-1-8	D BA8 1
㉒	11-1-8	D B 8 1
㉓	Ø-1-8	D A8 1
㉔	1-8	D 8 1
㉕	Ø-2-8-9	DCBA
㉖	12-11	D BA

Bedeutung der Gerätesteuerzeichen (Funktionszeichen)

Abkürzung	Funktion	
ACK	Acknowledge	Empfangsbestätigung
BEL	Bell	Akustisches Signal (Klingel)
BS	Backspace	Rücksetzen
BYP	Bypass	Übergehen
CAN	Cancel	Aufhebung
CC	Cursor control	Steuerung der Positionsanzeige
CR	Carriage return	Schreibkopf- oder Wagenrücklauf
CU1	Customer use 1	Kundenbenutzung 1
CU2	Customer use 2	Kundenbenutzung 2
CU3	Customer use 3	Kundenbenutzung 3
DC1	Device control 1	Gerätesteuerung 1
DC2	Device control 2	Gerätesteuerung 2
DC4	Device control 4	Gerätesteuerung 4
DEL	Delete	Löschen
DLE	Data link escape	Umschaltung von Text auf Steuerzeichen
DS	Digit select	Ziffernauswahl
EM	End of medium	Mediumende (Datenträgerende)
ENQ	Enquiry	Anfrage
EOT	End of transmission	Ende der Übertragung
ESC	Escape	Auswahl, Trennung, Umschaltung
ETB	End of transmission block	Ende eines Übertragungsblockes
ETX	End of text	Textende
FF	Form feed	Formularvorschub
FS	Field separator	Feldtrennzeichen
GE	Graphic escape	Graphische Auswahl, Trennung
HT	Horizontal tab	Horizontaleinstellung (Tabulator)
IFS	Interchange file separator	Schnittstellen-Dateitrennung
IGS	Interchange group separator	Schnittstellen-Gruppentrennung
IL	idle	Leerlauf
IRS	Interchange record separator	Schnittstellen-Satztrennung
IUS	Interchange unit separator	Schnittstellen-Einheitentrennung
LC	Lower case	Kleinbuchstabenschreibung
LF	Line feed	Zeilenvorschub
NAK	Negative achnowledge	Negative Empfangsbestätigung
NL	New line	Neue Zeile
NUL	Null	Nichtiges (ungültiges) Zeichen
PF	Punch off	Stanzen (Lochen) aus
PN	Punch on	Stanzen (Lochen) ein
RES	Restore	Rückspeichern
RLF	Reverse line feed	Umgekehrter Zeilenvorschub
RS	Reader stop	Leserstop
SI	Shift in	Umschaltung ein (Großbuchstabenschreibung)
SM	Set mode	Modus setzen
SMM	Start of manual message	Beginn einer manuellen Nachricht
SO	Shift out	Umschaltung aus (Kleinbuchstabenschreibung)
SOH	Start of heading	Beginn des Kopfes (Titels)
SOS	Start of significance	Start geltender Ziffern (Zeichen)
SP	Space	Leerstelle
STX	Start of text	Textanfang
SUB	Substitute	Ersetzen
SYN	Synchronous idle	Synchronisierzeichen (Leerlauf)
TM	Tape mark	Bandmarke (Bandbegrenzung)
UC	Upper case	Umschaltung auf Großbuchstaben
VT	Vertical tab	Vertikaleinstellung (Tabulator)

4. ASCII

ASCII ist die Abkürzung von *American Standard Code for Information Interchange (Amerikanischer Standardcode für den Informationsaustausch).*

Zweite Sedezimalziffer	Erste Sedezimalziffer: Ø	1	2	3	4	5	6	7
Ø	NUL	DLE	SP	Ø	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	¬	n	~
F	SI	US	/	?	O	-	o	DEL

Bedeutung der Gerätesteuerzeichen

Mit Ausnahme der folgenden haben die aufgeführten Abkürzungen die gleiche Bedeutung wie beim EBCDI-Code.

Abkürzung	Funktion	
GS	Group separator	Gruppentrennzeichen
RS	Record separator	Satztrennzeichen
US	Unit separator	Trennzeichen für eine Informationseinheit

Anhang B: Tabelle aller Instruktionen der IBM /370

E	For-mat	OP	Mnemoni-scher OP	Instruktion	BS	Programm-Ausnahmebedingung																				
						1	2	3	4s	4f	5	6	7	8	9	A	B	C	D	E	F	1Ø	11	12	13	4Ø
				Undefined OP		S																				
Ø	RR	Ø4	SPM	Set Program Mask	Ø123																					
Ø	RR	Ø5	BALR	Branch and Link																						
Ø	RR	Ø6	BCTR	Branch on Count																						
Ø	RR	Ø7	BCR	Branch on Condition																						
Ø	RR	Ø8	SSK	Set Storage Key			S				T	S														
Ø	RR	Ø9	ISK	Insert Storage Key			S				T	S														
Ø	RR	ØA	SVC	Supervisor Call																						
Ø	RR	ØE	MVCL	Move Long	Ø123				T	T	T	S										N	N	S		
Ø	RR	ØF	CLCL	Compare Logical Long	Ø12					T	T	S											N	S		
Ø	RR	1Ø	LPR	Load Positive	Ø 23									C												
Ø	RR	11	LNR	Load Negative	Ø12																					
Ø	RR	12	LTR	Load and Test	Ø12																					
Ø	RR	13	LCR	Load Complement	Ø123									C												
Ø	RR	14	NR	And	Ø1																					
Ø	RR	15	CLR	Compare Logical	Ø12																					
Ø	RR	16	OR	Or	Ø1																					
Ø	RR	17	XR	Exclusive Or	Ø1																					
Ø	RR	18	LR	Load																						
Ø	RR	19	CR	Compare	Ø12																					
Ø	RR	1A	AR	Add	Ø123									C												
Ø	RR	1B	SR	Subtract	Ø123									C												
Ø	RR	1C	MR	Multiply								S														
Ø	RR	1D	DR	Divide								S			S											
Ø	RR	1E	ALR	Add Logical	Ø123																					
Ø	RR	1F	SLR	Subtract Logical	123																					
2	RR	2Ø	LPDR	Load Positive Long	Ø 2	S						S														
2	RR	21	LNDR	Load Negative Long	Ø1	S						S														
2	RR	22	LTDR	Load and Test Long	Ø12	S						S														
2	RR	23	LCDR	Load Complement Long	Ø12	S						S														
2	RR	24	HDR	Halve Long		S						S							C							
3	RR	25	LRDR	Load Rounded Extended to Long		S						S						C								
3	RR	26	MXR	Multiply Extended		S						S						C	C							
3	RR	27	MXDR	Multiply Long to Extended		S						S						C	C							
2	RR	28	LDR	Load Long		S						S														
2	RR	29	CDR	Compare Long	Ø12	S						S														
2	RR	2A	ADR	Add Normalized Long	Ø12	S						S						C	C	C						
2	RR	2B	SDR	Subtract Normalized Long	Ø12	S						S						C	C	C						
2	RR	2C	MDR	Multiply Long		S						S						C	C							
2	RR	2D	DDR	Divide Long		S						S						C	C		S					
2	RR	2E	AWR	Add Unnormalized Long	Ø12	S						S						C		C						
2	RR	2F	SWR	Subtract Unnormalized Long	Ø12	S						S						C		C						
2	RR	3Ø	LPER	Load Positive Short	Ø 2	S						S														
2	RR	31	LNER	Load Negative Short	Ø1	S						S														
2	RR	32	LTER	Load and Test Short	Ø12	S						S														
2	RR	33	LCER	Load Complement Short	Ø12	S						S														
2	RR	34	HER	Halve Short		S						S							C							
2	RR	35	LRER	Load Rounded Short		S						S						C								
2	RR	36	AXR	Add Normalized Extended	Ø12	S						S						C	C	C						
2	RR	37	SXR	Subtract Normalized Extended	Ø12	S						S						C	C	C						
2	RR	38	LER	Load Short		S						S														
2	RR	39	CER	Compare Short	Ø12	S						S														
2	RR	3A	AER	Add Normalized Short	Ø12	S						S						C	C	C						
2	RR	3B	SER	Subtract Normalized Short	Ø12	S						S						C	C	C						
2	RR	3C	MER	Multiply Short		S						S						C	C							
2	RR	3D	DER	Divide Short		S						S						C	C		S					
2	RR	3E	AUR	Add Unnormalized Short	Ø12	S						S						C		C						
2	RR	3F	SUR	Subtract Unnormalized Short	Ø12	S						S						C		C						

						Programm-Ausnahmebedingung																									
E	Format	OP	Mnemonischer OP	Instruktion	BS	1	2	3	4s	4f	5	6	7	8	9	A	B	C	D	E	F	1Ø	11	12	13	4Ø
Ø	RX	4Ø	STH	Store Halfword					T		T											N	N	S		
Ø	RX	41	LA	Load Address																						
Ø	RX	42	STC	Store Character					T		T											N	N	S		
Ø	RX	43	IC	Insert Character						T	T											N	N	S		
Ø	RX	44	EX	Execute				S		T	T	S										N	N	S		
Ø	RX	45	BAL	Branch and Link																						
Ø	RX	46	BCT	Branch and Count																						
Ø	RX	47	BC	Branch on Condition																						
Ø	RX	48	LH	Load Halfword						T	T											N	N	S		
Ø	RX	49	CH	Compare Halfword	Ø12					T	T											N	N	S		
Ø	RX	4A	AH	Add Halfword	Ø123					T	T			C								N	N	S		
Ø	RX	4B	SH	Subtract Halfword	Ø123					T	T			C								N	N	S		
Ø	RX	4C	MH	Multiply Halfword						T	T											N	N	S		
Ø	RX	4E	CVD	Convert to Decimal					T		T											N	N	S		
Ø	RX	4F	CVB	Convert to Binary						T	T		S			C						N	N	S		
Ø	RX	5Ø	ST	Store					T		T											N	N	S		
Ø	RX	54	N	And	Ø1					T	T											N	N	S		
Ø	RX	55	CL	Compare Logical	Ø12					T	T											N	N	S		
Ø	RX	56	O	Or	Ø1					T	T											N	N	S		
Ø	RX	57	X	Exclusive Or	Ø1					T	T											N	N	S		
Ø	RX	58	L	Load						T	T											N	N	S		
Ø	RX	59	C	Compare	Ø12					T	T											N	N	S		
Ø	RX	5A	A	Add	Ø123					T	T			C								N	N	S		
Ø	RX	5B	S	Subtract	Ø123					T	T			C								N	N	S		
Ø	RX	5C	M	Multiply						T	T	S										N	N	S		
Ø	RX	5D	D	Divide						T	T	S			S							N	N	S		
Ø	RX	5E	AL	Add Logical	Ø123					T	T											N	N	S		
Ø	RX	5F	SL	Subtract Logical	123					T	T											N	N	S		
2	RX	6Ø	STD	Store Long		S			S		S	S										N	N	S		
3	RX	67	MXD	Multiply Long to Extended		S				T	T	S							C	C		N	N	S		
2	RX	68	LD	Load Long		S				T	T	S										N	N	S		
2	RX	69	CD	Compare Long	Ø12	S				T	T	S										N	N	S		
2	RX	6A	AD	Add Long	Ø12	S				T	T	S						C	C	C		N	N	S		
2	RX	6B	SD	Subtract Long	Ø12	S				T	T	S						C	C	C		N	N	S		
2	RX	6C	MD	Multiply Long		S				T	T	S						C	C			N	N	S		
2	RX	6D	DD	Divide Long		S				T	T	S						C	C		S	N	N	S		
2	RX	6E	AW	Add Unnormalized Long	Ø12	S				T	T	S						C		C		N	N	S		
2	RX	6F	SW	Subtract Unnormalized Long	Ø12	S				T	T	S						C		C		N	N	S		
2	RX	7Ø	STE	Store Short		S			T		T	S										N	N	S		
2	RX	78	LE	Load Short		S				T	T	S										N	N	S		
2	RX	79	CE	Compare Short	Ø12	S				T	T	S										N	N	S		
2	RX	7A	AE	Add Normalized Short	Ø12	S				T	T	S						C	C	C		N	N	S		
2	RX	7B	SE	Subtract Normalized Short	Ø12	S				T	T	S						C	C	C		N	N	S		
2	RX	7C	ME	Multiply Short		S				T	T	S						C	C			N	N	S		
2	RX	7D	DE	Divide Short		S				T	T	S						C	C		S	N	N	S		
2	RX	7E	AU	Add Unnormalized Short	Ø12	S				T	T	S						C		C		N	N	S		
2	RX	7F	SU	Subtract Unnormalized Short	Ø12	S				T	T	S						C		C		N	N	S		

						Programm-Ausnahmebedingung																				
E	Format	OP	Mnemonischer OP	Instruktion	BS	1	2	3	4s	4f	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	4Ø
Ø	SI	8Ø	SSM	Set System Mask			S			S	S								S			N	N	S	S	
Ø	SI	82	LPSW	Load Program Status Word	Ø123		S			S	S	S										N	N	S		
Ø	SI	83	-	Diagnose	Ø123	←				alle		→														
5	SI	84	WRD	Write Direct		S	S			T	T											N	N	S		
5	SI	85	RDD	Read Direct		S	S		T		T											N	N	S		
Ø	RS	86	BXH	Branch on Index High																						
Ø	RS	87	BXLE	Branch on Index Low or Equal																						
Ø	RS	88	SRL	Shift Right Single Logical																						
Ø	RS	89	SLL	Shift Left Single Logical																						
Ø	RS	8A	SRA	Shift Right Single Arithmetic	Ø12																					
Ø	RS	8B	SLA	Shift Left Single Arithmetic	Ø123									C												
Ø	RS	8C	SRDL	Shift Right Double Logical								S														
Ø	RS	8D	SLDL	Shift Left Double Logical								S														
Ø	RS	8E	SRDA	Shift Right Double Arithmetic	Ø12							S														
Ø	RS	8F	SLDA	Shift Left Double Arithmetic	Ø123							S		C												
Ø	RS	9Ø	STM	Store Multiple					T		T											N	N	S		
Ø	SI	91	TM	Test under Mask	Ø123					T	T											N	N	S		
Ø	SI	92	MVI	Move Immediate					T		T											N	N	S		
Ø	SI	93	TS	Test and Set	Ø1				T	T	T											N	N	S		
Ø	SI	94	NI	And Immediate	Ø1				T	T	T											N	N	S		
Ø	SI	95	CLI	Compare Logical Immediate	Ø12					T	T											N	N	S		
Ø	SI	96	OI	Or Immediate	Ø1				T	T	T											N	N	S		
Ø	SI	97	XI	Exclusive Or Immediate	Ø1				T	T	T											N	N	S		
Ø	RS	98	LM	Load Multiple						T	T											N	N	S		
Ø	S	9CØØ	SIO	Start I/Ø	Ø123		S																			
Ø	S	9CØ1	SIOF	Start I/Ø Fast Release	Ø123		S																			
Ø	S	9DØØ	TIO	Test I/Ø	Ø123		S																			
Ø	S	9DØ1	CLRIO	Clear I/Ø	Ø123		S																			
Ø	S	9EØØ	HIO	Halt I/Ø	Ø123		S																			
Ø	S	9EØ1	HDV	Halt Device	Ø123		S																			
Ø	S	9FØØ	TCH	Test Channel	Ø123		S																			
6	SI	AC	STNSM	Store then and System Mask		S	S		S		S											N	N	S		
6	SI	AD	STOSM	Store then or System Mask		S	S		S		S											N	N	S		
7	RS	AE	SIGP	Signal Processor	Ø123	S	S																			
Ø	SI	AF	MC	Monitor Call								S														C
6	RX	B1	LRA	Load Real Address	Ø123	S	S															N	N	S		
Ø	S	B2Ø2	STIDP	Store CPU ID			S		T		T	S										N	N	S		
Ø	S	B2Ø3	STIDC	Store Channel Id	Ø123		S																			
Ø	S	B2Ø4	SCK	Set Clock	Ø1 3		S			T	T	S										N	N	S		
Ø	S	B2Ø5	STCK	Store Clock	Ø123				T		T											N	N	S		
4	S	B2Ø6	SCKC	Set Clock Comparator		S	S		S		S	S										N	N	S		
4	S	B2Ø7	STCKC	Store Clock Comparator		S	S		T		T	S										N	N	S		
4	S	B2Ø8	SPT	Set CPU Timer		S	S			S	S	S										N	N	S		
4	S	B2Ø9	STPT	Store CPU Timer		S	S		T		T	S										N	N	S		
9	S	B2ØA	SPKA	Set PSW Key from Address		S	S																			
9	S	B2ØB	IPK	Insert PSW Key		S	S																			
6	S	B2ØD	PTLB	Purgo TLB		S	S																			
7	S	B21Ø	SPX	Set Prefix		S	S	S			S	S										N	N	S		
7	S	B211	STPX	Store Prefix		S	S		T		T	S										N	N	S		
7	S	B212	STAP	Store CPU Address		S	S		T		T	S										N	N	S		
6	S	B213	RRB	Reset Reference Bit	Ø123	S	S				T															
Ø	RS	B6	STCTL	Store Control			S		T		T	S										N	N	S		
Ø	RS	B7	LCTL	Load Control			S		T		T	S										N	N	S		
8	RS	BA	CS	Compare and Swap	Ø1	S			T	T	T	S										N	N	S		
8	RS	BB	CDS	Compare Double and Swap	Ø1	S			T	T	T	S										N	N	S		
Ø	RS	BD	CLM	Compare Logical Characters under Mask	Ø12					T	T											N	N	S		
Ø	RS	BE	STCM	Store Characters under Mask					T		T											N	N	S		
Ø	RS	BF	ICM	Insert Characters under Mask	Ø12					T	T											N	N	S		

						Programm-Ausnahmebedingung																					
E	Format	OP	Mnemonischer OP	Instruktion	BS	1	2	3	4s	4f	5	6	7	8	9	A	B	C	D	E	F	1Ø	11	12	13	4Ø
Ø	SS	D1	MVN	Move Numerics					T	T	T											N	N	S		
Ø	SS	D2	MVC	Move Characters					T	T	T											N	N	S		
Ø	SS	D3	MVZ	Move Zones					T	T	T											N	N	S		
Ø	SS	D4	NC	And Characters	Ø1				T	T	T											N	N	S		
Ø	SS	D5	CLC	Compare Logical Characters	Ø12					T	T											N	N	S		
Ø	SS	D6	OC	Or Characters	Ø1				T	T	T											N	N	S		
Ø	SS	D7	XC	Exclusive Or Characters	Ø1				T	T	T											N	N	S		
Ø	SS	DC	TR	Translate					T	T	T											N	N	S		
Ø	SS	DD	TRT	Translate and Test	Ø12					T	T											N	N	S		
1	SS	DE	ED	Edit	Ø12	S			T	T	T		T									N	N	S		
1	SS	DF	EDMK	Edit and Mark	Ø12	S			T	T	T		T									N	N	S		
1	SS	FØ	SRP	Shift and Round Decimal	Ø123	S			T	T	T		T			C						N	N	S		
Ø	SS	F1	MVO	Move with Offset					T	T	T											N	N	S		
Ø	SS	F2	PACK	Pack					T	T	T											N	N	S		
Ø	SS	F3	UNPK	Unpack					T	T	T											N	N	S		
2	SS	F8	ZAP	Zero and Add	Ø123	S			T	T	T		S			C						N	N	S		
2	SS	F9	CP	Compare Decimal	Ø12	S				T	T		S									N	N	S		
2	SS	FA	AP	Add Decimal	Ø123	S			T	T	T		S			C						N	N	S		
2	SS	FB	SP	Subtract Decimal	Ø123	S			T	T	T		S			C						N	N	S		
2	SS	FC	MP	Multiply Decimal		S			T	T	T	S	S									N	N	S		
2	SS	FD	DP	Divide Decimal		S			T	T	T	S	S				S					N	N	S		

Anmerkungen:

1) Unter E ist vermerkt, ob die betreffende Instruktion zur Standardausrüstung der IBM /370 gehört oder zu einer Zusatzeinrichtung. Es bedeuten:

0	Standardausrüstung
1	Einrichtung für die Dezimalarithmetik
2	Gleitpunkteinrichtung
3	Erweiterte Gleitpunkteinrichtung
4	Zeitgeber in der Zentraleinheit und Uhrzeitvergleicher
5	Direktsteuerung
6	Übersetzung
7	Multiprocessing (Simultanverarbeitung)
8	Bedingter Austausch
9	Behandlung des PSW-Schlüssels

2) Unter BS sind die Werte aufgeführt, die der Bedingungsschlüssel annehmen kann

3) In den Spalten „Programm-Ausnahmebedingungen" haben die einzelnen Spalten bzw. die in ihnen verwendeten Abkürzungen folgende Bedeutungen:

a) Bedeutung der Spalten

Spalte	Bedeutung
1	Ungültiger Operationsschlüssel
2	Privilegierte Operation im Problemstatus
3	Fehler bei der Instruktion „Ausführung"
4s	Speicherschutzfehler: Schreibsperre (Hauptspeicherschutz)
4f	Speicherschutzfehler: Lesesperre (Abrufschutz)
5	Adressierungsfehler
6	Spezifikationsfehler
7	Datenfehler
8	Überlauf bei der Festpunktarithmetik
9	Überlauf bei der Festpunktdivision
A	Dezimalüberlauf
B	Dezimal-Divisionsüberlauf
C	Exponentenüberlauf
D	Exponentenunterlauf
E	Gültigkeitsfehler
F	Gleitpunktdivisionsüberlauf
10	Segmentumsetzungsfehler
11	Seitenumsetzungsfehler
12	Spezifikationsfehler bei der Umsetzung
13	Fehler bei speziellen Operationen
40	Monitorereignis

b) Bedeutung der Spalteneintragungen

Abkürzung	Bedeutung
S	Die Operation wird beim Auftreten der Bedingung unterdrückt
T	Die Operation wird beim Auftreten der Bedingung abgebrochen
C	Die Operation wird beendet
N	Die Operation wird nullifiziert

4) Während der Interpretationsphase einer Instruktion (Einlesen der Instruktion vom Hauptspeicher ins Steuerwerk) können folgende Ausnahmebedingungen erkannt werden:

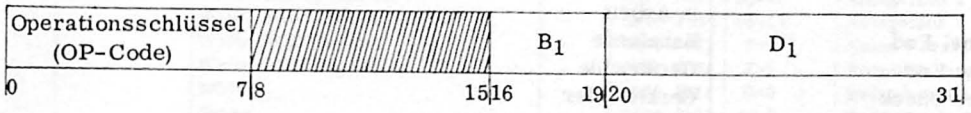
a) Hauptspeicher-Lesesperre (S)
b) Adressierungsfehler (S)
c) Segmentumsetzungsfehler (N)
d) Seitenumsetzungsfehler (N)
e) Spezifikationsfehler bei der Umsetzung (S)

5) Die deutsche Übersetzung der Operationsschlüssel ist den entsprechenden Kapiteln im Buch zu entnehmen.

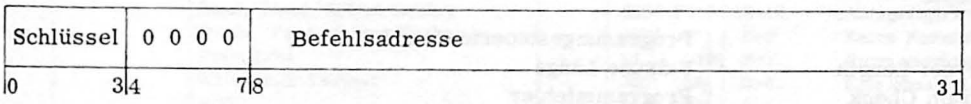
Anhang C: Übersicht über die Eingabe/Ausgabe

1. Instruktionen, Befehle und Steuerwörter

a) Eingabe/Ausgabe – Instruktionen

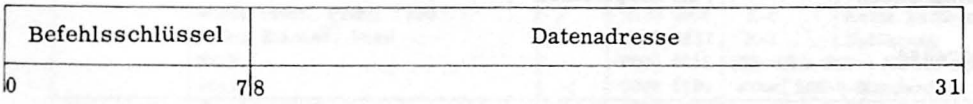


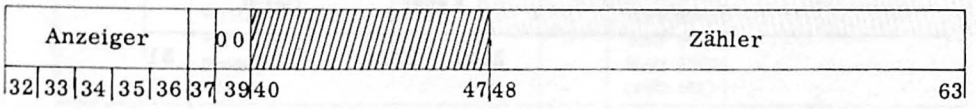
b) Kanaladreßwort (Channel Address Word —— CAW)



Das Kanaladreßwort besitzt die Hauptspeicheradresse 72 bzw. 48 (sedezimal).

c) Kanalbefehlswort (Channel Command Word —— CCW)

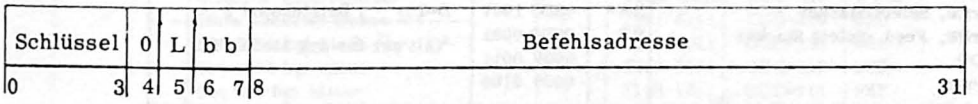


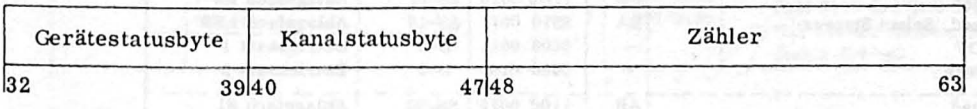


Die Anzeiger haben folgende Bedeutung:

Bitstelle 32: Datenkettung (Chain Data —— CD)
Bitstelle 33: Befehlskettung (Chain Command —— CC)
Bitstelle 34: Unterdrückung der Längenanzeige
Bitstelle 35: Keine Übertragung
Bitstelle 36: Programmgesteuerte Unterbrechung (Program Control Interrupt —— PCI)
Bitstelle 37: Indirekte Datenadresse

d) Kanalstatuswort (Channel Status Word ——CSW)





Das Kanalstatuswort besitzt die Hauptspeicheradresse 64 bzw. 40 (sedezimal).

Die Bitstellen haben folgende Bedeutung:

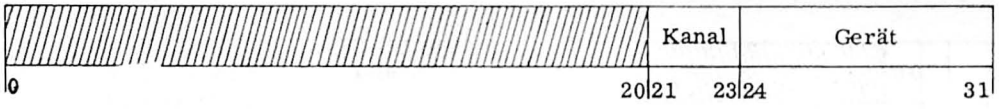
— Bitstelle 5: Hängender Logout
— Bitstellen 6 und 7: Zurückgestellter Bedingungsschlüssel
— Gerätestatusbyte

32	Attention	Achtung
33	Status Modifier	Statusänderung
34	Control Unit End	Steuereinheitenende
35	Busy	In Arbeit
36	Channel End	Kanalende
37	Device End	Geräteende
38	Device Check	Gerätefehler
39	Device Exception	Geräte-Ausnahmebedingung

— Kanalstatusbyte

40	PCI	Programmgesteuerte Unterbrechung
41	Incorrect Length	Falsche Länge
42	Program Check	Programmfehler
43	Protection Check	Speicherschutzfehler
44	Channel Data Check	Kanaldatenfehler
45	Channel Control Check	Kanalsteuerfehler
46	Interface Control Check	Interface-Steuerfehler
47	Chaining Check	Kettungsfehler

e) Geräteadresse



2. Befehlsschlüssel (ohne SEDZ)

Gerät	*Befehl*	*Befehls-typ*	*Schlüssel*	*Bemerkungen*	
IBM 2501	Read	-	00D0 0010	D=0	Betriebsart 1
	NOP	-	0000 0011	D=1	Betriebsart 2
	Sense	-	0000 0100		
IBM 2520	Read, Feed, Select Stacker	AA*	0SD0 0010	S=0	Ablagefach 1
	Read, Feed	AB*	11D0 0010	S=1	Ablagefach 2
	Select Stacker	BA*	0S10 0011	D=0	Betriebsart 1
	Write, Select Stacker	BA*	0SD0 1001	D=1	Betriebsart 2
	Write, Feed, Select Stacker	BB	0SD0 0001	*Gilt nur für das Modell B1	
	NOP	-	0000 0011		
	Sense	-	0000 0100		
IBM 2540 Leser	Read, Feed, Select Stacker	AA	SSD0 0010	SS=00	Ablagefach L1
	Read	AB	11D0 0010	SS=01	Ablagefach L2
	Feed, Select Stacker	BA	SS10 0011	SS=10	Ablagefach LS3
	NOP	-	0000 0011	D=0	Betriebsart 1
	Sense	-	0000 0100	D=1	Betriebsart 2
IBM 2540 Stanzer	Read	AB	1100 0010	SS=00	Ablagefach S1
	PFR, Write, Feed, Select Stacker [1]	BA	SSD0 1001	SS=01	Ablagefach S2
	Write, Feed, Select Stacker	BB	SSD0 0001	SS=10	Ablagefach LS3
	NOP	-	0000 0011	D=0	Betriebsart 1
	Sense	-	0000 0100	D=1	Betriebsart 2

Gerät	*Befehl*	*Befehls-typ*	*Schlüssel*	*Bemerkungen*	
IBM 2596	Read, Feed, Select Stacker	-	0S00 0010	S=0	Ablagefach 1 oder 3
	Select Stacker, Feed	-	FS00 0011	S=1	Ablagefach 2 oder 4
	Write, Feed, Select Stacker	-	1S00 0001	F=0	Keine Zuführung
	NOP	-	0000 0011	F=1	Zuführung
	Sense	-	0000 0100		
IBM 3504 IBM 3505	Read	-	11D0 F010	SS=00	Ablagefach 1
	Feed, Select Stacker	-	SS10 F011	SS=01	Ablagefach 2
	Read, Feed, Select Stacker	-	SSD0 F010	SS=10	Ablagefach 2
	Write RCE Format	-	0001 0001	F=0	Keine Formatierung
	Write OMR Format	-	0011 0001	F=1	Formatierung
	NOP	-	0000 0011	D=0	Betriebsart 1
	Sense	-	0000 0100	D=1	Betriebsart 2
IBM 3525	Read	-	11D0 F010	SS=00	Ablagefach 1
	Feed, Select Stacker	-	SS10 F011	SS=01	Ablagefach 2
	Read, Feed, Select Stacker	-	SSD0 F010	SS=10	Ablagefach 2
	Write, Feed, Select Stacker	-	SSD0 0001	F=0	Keine Formatierung
	Print Line	-	LLLL L101	F=1	Formatierung
	Write RCE Format	-	0001 0001	D=0	Betriebsart 1
	NOP	-	0000 0001	D=1	Betriebsart 2
	Sense	-	0000 0100	L=1 bis L=25	Zeilennummer
IBM 5425	Read, Feed	-	H0D0 0010	D=0	96-Spalten-Code
	Write Punch	-	H000 0101	D=1	IPL-Code
	Write Punch, Punch, Feed	-	H000 0001	H=0	Erstmagazin
	Write Print	-	H100 0101	H=1	Zweitmagazin
	Write Print, Print, Feed	-	H100 0001	F=0	Keine Zuführung
	Select Stacker, Feed	-	HSSS 0F11	F=1	Zuführung
	NOP	-	0000 0011	SSS=0 bis SSS=4 für Ablagefach,	
	Sense	-	0000 0100	wobei SSS=0 Standard ist	
IBM 3210 IBM 2315	Read Inquiry	-	0000 1010		
	Write and Carrier Return	-	0000 1001		
	Write, No Carrier Return	-	0000 0001		
	NOP	-	0000 0011		
	Sense	-	0000 0100		
	Alarm	-	0000 1011		
IBM 1403 IBM 1443 IBM 2303 IBM 3211 IBM 5203	Write and Space	-	000L L001	Anzahl der Zeilen-transporte	Vorschub bis zu einer Lochung im Kanal
	Space Immediately	-	000L L011		
	Write and Skip	-	1CCC C001		
	Skip Immediately	-	1CCC C001	LL=00	CCCC=0001
	Block Data Check	-	1CCC C011	01	bis
	Allow Data Check	-	0111 0011	10	CCCC=1100
	Load FCB	-	0111 1011	11	
	Read FCB	-	0110 0011		
	NOP	-	0001 0010		
	Sense	-	0000 0100		
Magnet-band-geräte	Read	-	0000 0000	CCC=000	REW
	Read Backwards	-	0000 1100	CCC=001	RUN
	Write	-	0000 1100	CCC=010	ERG
	Control Tape Motion	-	00CC C111	CCC=011	WTM
	Data Security Erase [1]	-	1001 0111	CCC=100	BSB
	Set 6250 bpi Mode [2]	-	1101 0011	CCC=101	BSF
	Set 1600 bpi Mode	-	1100 0011	CCC=110	FSB
	Set 800 bpi Mode	-	1100 1011	CCC=111	FSF
	Request TIE [3]	-	1101 1011	1) Nur bei den Typen IBM 3410 und IBM 3420	
	NOP	-	0000 0011	2) Nur bei der Aufzeichnungsdichte 800 bpi	
	Sense	-	0000 0100		

Es bedeuten: 1) PFR — Punch, Feed, Read (Stanzen in gleiche Lochkarte)
2) bpi — Bytes per inch (Bytes/Zoll)
3) TIE — Track in Error (Spurfehler)

Die deutschen Bezeichnungen der Befehle können dem Kap. 6 entnommen werden.

3. Befehlsschlüssel bei SEDZ

Befehl	Zähler	Schlüssel	Bemerkungen	
Seek	6	0000 0111		
Seek Cylinder	6	0000 1010		
Seek Head	6	0001 1010		
Read Sector	1	0010 0010	} nicht bei IBM 2311, IBM 2314 und IBM 2319	
Set Sector	1	0010 0011		
Search HA Equal [1]	4	M011 1001	CC=01	Gleich
Search ID [2]	5	MCC1 0001	CC=10	Hoch
Search Key	[5]	MCC0 1001	CC=11	Gleich oder hoch
Search Key and Data	[5]	MCC0 1101		
Space Count	3	0000 1111		
Set File Mask	1	0001 1111	M=0	Einspuroperation
Erase	-	0001 0001	M=1	Mehrspuroperation
Read	[5]	M00D DD10	DDD=001	Daten
Read Count	8	M001 0010	DDD=011	Schlüssel und Daten
Read IPL [3]	24	0000 0010	DDD=101	Spurbeschreibungssatz
Write	[5]	000D DD01	DDD=110	Spuradresse
Write Special Count, Key, and Data	[5]	0000 0001	DDD=111	Kennzeichnung, Schlüssel und Daten
Recalibrate	-	0001 0011		
Restore	-	0001 0111		
NOP [4]	-	0000 0011		
Tic — Transfer in Channel	-	xxxx 1000		
Sense	4	0000 0100		
Device Restore	-	1001 0100		
Device Reserve	-	1011 0100		

Es bedeuten:
1) HA — Home Address (Spuradresse oder auch Hausadresse)
2) ID — Identification (Kennzeichnung)
3) IPL — Initial Program Loading (Einleitendes Programmladen)
4) NOP — No Operation (Keine Operation)
5) Die Anzahl der in die Operation eingehenden Bytes ist durch den Zähler im CCW festgelegt.

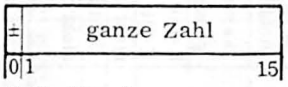
Die deutschen Bezeichnungen der Befehle können dem Kap. 6 entnommen werden.

Anhang D: Referenzdaten

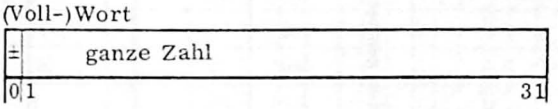
1. Datenformate

a) Dualzahlen in Festpunktdarstellung

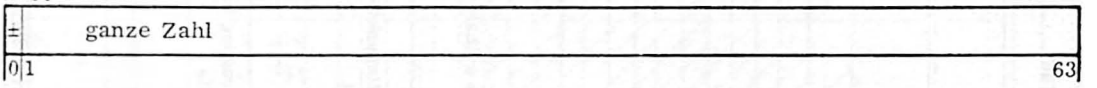
Halbwort



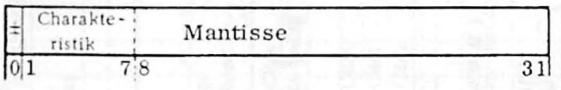
(Voll-)Wort

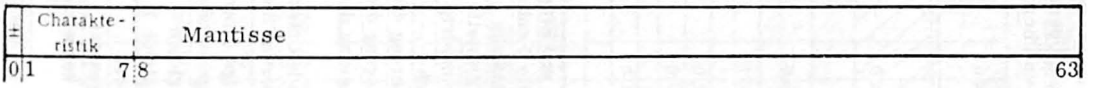


Doppelwort



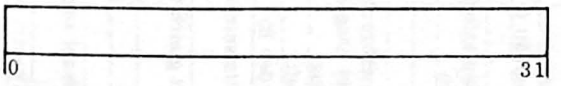
b) Dualzahlen in Gleitpunktdarstellung



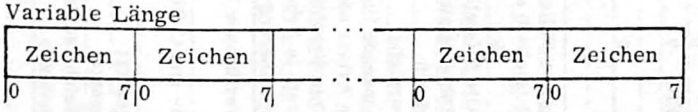


c) Logische Daten

(Voll-)Wort

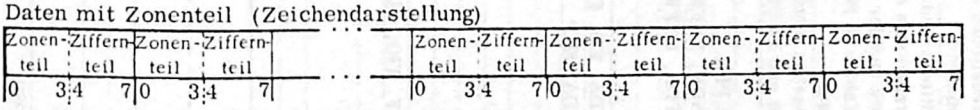


Variable Länge

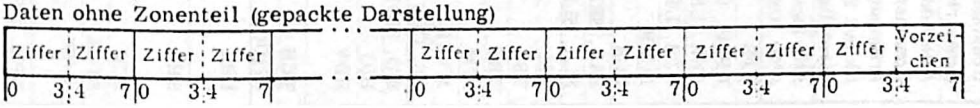


d) Dezimale Daten

Daten mit Zonenteil (Zeichendarstellung)



Daten ohne Zonenteil (gepackte Darstellung)



2. Werte des Bedingungsschlüssels (BS)

Instruktionen	*Werte des Bedienungsschlüssels*			
	0	1	2	3
Alle Vergleichsinstruktionen	Gleichheit der Operanden	Operand 1 < Operand 2	Operand 1 > Operand 2	-
AL, ALR	0, kein Übertrag	≠ 0, kein Übertrag	0, kein Übertrag	0, Übertrag
SL, SLR	-	≠ 0, kein Übertrag	0	0, Übertrag
Alle Additions- und Subtraktionsinstruktionen der Gleitpunktarithmetik	0	< 0	> 0	-
Alle Additions-und Subtraktionsinstruktionen der Dezimalarithmetik und der Dualarithmetik mit Zahlen in Festpunktdarstellung	0	< 0	> 0	Überlauf
Alle logischen Instruktionen AND, OR und EXCLUSIVE OR	0	≠ 0	-	-
LPR	0	-	> 0	Überlauf
LPDR, LPER	0	-	> 0	-
LNR, LNDR, LNER	0	< 0	-	-
LTR, LTDR, LTER, LCDR, LCER	0	< 0	> 0	-
LCER				-
LCR	0	< 0	> 0	Überlauf
SRA, SRDA, ED, EDMK				
SLA, SLDA, SRP				
TM	lauter Nullen	Nullen und Einsen	-	lauter Einsen
TS	Bitstelle 0 enthält 0	Bitstelle 0 enthält 1	-	-
TRT	0	unvollständig	vollständig	-
ICM	Alle Bitstellen enthalten 0	Erste Bitstelle enthält 1	Erste Bitstelle enthält 0	-
MVCL	Zähler gleich	Zähler niedrig	Zähler hoch	Überlappung
CS, CDS	gleich	ungleich	-	-
SIO, SIOF, TIO, CLRIO	verfügbar	Speicherung des CSW	in Arbeit	nicht betriebsbereit
HIO	hängende Unterbrechung	Speicherung des CSW	Operation im Einpunktbetrieb gestoppt	nicht betriebsbereit
HDV	in Arbeit oder hängende Unterbrechung	Speicherung des CSW	Kanal in Arbeit	nicht betriebsbereit
TCH	nicht in Arbeit	hängende Unterbrechung	in Arbeit	nicht betriebsbereit
STIDC	Speicherung der ID	Speicherung des CSW	in Arbeit	nicht betriebsbereit
LRA	Verfügbare Umsetzung	Ungültige Eintragung in die Segmenttabelle	Ungültige Eintragung in die Seitentabelle	Verletzung der Längenangabe
RRB	R- und C-Bit gleich 0	R-Bit gleich 0 C-Bit gleich 1	R-Bit gleich 1 C-Bit gleich 0	R-Bit gleich 1 C-Bit gleich 1
SCK	gesetzt	gesichert	-	nicht betriebsbereit
STCK	gesetzt	nicht gesetzt	Fehler	nicht betriebsbereit
SIGP	Befehlsschlüssel angenommen	Status gespeichert	in Arbeit	nicht betriebsbereit
SPM, LPSW	Gesetzt von der Programm-Maske im Programmstatuswort (PSW)			
Diagnose	kann jeden Bedienungsschlüssel setzen			

3. Modelle des Systems IBM/370 und ihre Kenngrößen

Modell-nummer	*Hauptspeicher-kapazität*	*Modelle*								
		115	125	135	145	155	158	165	168	195
F	65 536									
FE	98 3Ø4									
G	131 Ø72									
GD	147 456									
GE	163 84Ø									
GF	196 6Ø8									
GFD	212 992									
DH	245 76Ø									
H	262 144									
HF	327 68Ø									
HG	393 216									
I	524 288									
IH	786 432									
J	1 Ø48 576									
JI	1 572 864									
K	2 Ø97 152									
KJ	3 145 728									
L	4 194 3Ø4									
LJ	5 242 88Ø									
LK	6 291 456									
LKJ	7 34Ø Ø32									
M	8 388 6Ø8									
Durchschnittliche Ausführung der Instruktion mit den Operationsschlüssel 5A (Addieren) in Mikrosekunden		1447Ø	98ØØ	4Ø84	2385	993	933	29Ø	2Ø8	135

4. Programmstatuswort (PSW)

a) Basissteuerung (BC)

Kanalmasken 0 bis 5	EA	eU	Schlüssel		Unterbrechungsschlüssel
0 5	6	7	8 11	12 15	16 31

ILS	BS	Programm-Maske	Instruktionsadresse
32 33	34 35	36 39	40 63

Bitstellen	Bedeutung
0 bis 5	Masken für die Kanäle 0 bis 5
6	Maske für die Kanäle 6 bis 13 (EA)
7	Maske für externe Unterbrechungen (eU)
8 bis 11	Schutzschlüssel
12	0 (Basissteuerungskennung)
13	Maschinenfehler-Maske (M)
14	Wartestatus (W)
15	Problemstatus (P)
16 bis 31	Unterbrechungsschlüssel
32 und 33	Instruktionslängenschlüssel (ILS)
34 und 35	Bedingungsschlüssel (BS)
36 bis 39	Programm-Maske 36 Festpunktüberlauf-Maske 37 Dezimalüberlauf-Maske 38 Exponentenunterlauf-Maske 39 Gültigkeits-Maske
40 bis 63	Instruktionsadresse

b) Erweiterte Steuerung (EC)

0	PER	000	U	EA	eU	Schlüssel	1	MWP-Schalter	00	BS	Programm-maske	0 0 0 0 0 0 0 0
0	1	5	6	7	8	11	12	13 15		18 19	20 23	24 31

0 0 0 0 0 0 0 0	Instruktionsadresse
32 39	40 63

Bitstellen	Bedeutung
1	Programm-Ereignisregistrierung (PER)
5	Dynamische Adreßumsetzung (U)
6	Maske für alle Kanäle (EA)
7	Maske für externe Unterbrechungen (eU)
8 bis 11	Schutzschlüssel
12	1 (Kennung für die erweiterte Steuerung)
13	Maschinenfehler-Maske (M)
14	Wartestatus (W)
15	Problemstatus (P)
18 und 19	Bedingungsschlüssel (BS)
20 bis 23	Programm-Maske
	20 Festpunktüberlauf-Maske
	21 Dezimalüberlauf-Maske
	22 Exponentenunterlauf-Maske
	23 Gültigkeits-Maske
40 bis 63	Instruktionsadresse

Anmerkung: Alle nicht aufgeführten Bitstellen müssen Nullen enthalten

Eine 1 auf einer Maskenbitstelle läßt es zu einer Unterbrechung kommen, eine 0 verhindert sie.

Eine 1 auf einer Bitstelle des MWP-Feldes bewirkt, daß der Computer in die entsprechende Arbeitsweise versetzt wird. Eine 0 auf Bitstelle 13 bewirkt, daß Maschinenfehler unberücksichtigt bleiben, eine 0 auf 14, daß sich der Computer im Laufstatus befindet,und eine 0 auf 15, daß er sich im Supervisorstatus befindet.

5. Aufbereitung

Sedezimale Verschlüsselung	Bedeutung	Symbolisches Zeichen	Lochkartencode	Engl. Abk.
20	Ziffernauswahlzeichen	▽	11-0-1-8-9	DS
21	Startzeichen für geltende Ziffern	[	0-1-0	SOS
22	Feldtrennungszeichen	]	0-2-9	FS

6. Schlüssel für Programmunterbrechungen

Hexadezimaler Schlüssel	Ursache
0001	Operation (ungültige)
0002	Privilegierte Operation
0003	Ausführen (EX)
0004	Speicherschutz
0005	Adressierung
0006	Spezifikation
0007	Datenfehler
0008	Festpunkt-Überlauf
0009	Festpunkt-Divisionsüberlauf
000A	Dezimalüberlauf
000B	Dezimal-Divisionsüberlauf
000C	Exponentenüberlauf
000D	Exponentenunterlauf
000E	Gültigkeitsfehler
000F	Gleitpunkt-Divisionsüberlauf
0010	Segmentumsetzung
0011	Seitenumsetzung
0012	Umsetzungs-Ausnahmebedingung
0013	Spezielle Operation
0040	Monitorereignis
0080	Programmereignis

7. Steuerungsregister

Steuerungsregister	Bitstellen	Bedeutung	Anfangswert
0	0	Blockmultiplex-Modus	0
	1	Unterdrückung der Instruktion SSM	0
	8 und 9	Seitenumfang	0
	11 und 12	Segmentumfang	0
	20	Uhrzeitvergleicher-Maske	0
	21	Maske für internen Zeitgeber	0
	24	Maske für Intervallzeitgeber	1
	25	Maske für Unterbrechungstaste	1
	26	Maske für externes Signal	1
1	0 bis 7	Länge der Segmenttabelle	0
	8 bis 25	Adresse der Segmenttabelle	0
2	0 bis 31	Kanalmasken	1
3	16 bis 31	Monitormasken	0
9		Programm-Ereignisregistrierung (PER)	
	0	Maske für erfolgte Verzweigungen	0
	1	Maske für Instruktionsabruf	0
	2	Maske für Speicheränderung	0
	3	Maske für Änderung allgemeiner Register	0
	16 bis 31	Maske für die Identifizierung allgemeiner Register	0
10	0 bis 31	Startadresse für die Programm-Ereignisregistrierung (PER)	0
11	0 bis 31	Endadresse für die Programm-Ereignisregistrierung (PER)	0
14	0	Fehlerstop-Steuerung bei Maschinenfehlern	1
	1	Synchrone Steuerung von erweiterten Logouts bei Maschinenfehlern	1
	2	Steuerung von erweiterten Logouts bei der EA	0
	4	Maske für die Berichte über die Beseitigung von Maschinenfehlern	0
	5	Maske für die Berichte über die Absetzung von Maschinenfehlern	0
	6	Maske für die Berichte über externe Schäden (Maschinenfehler)	1
	7	Maske für Warnungen über Maschinenfehler	0
	8	Asynchrone Steuerung von erweiterten Logouts bei Maschinenfehlern	0
	9	Asynchrone Steuerung von festen Logouts bei Maschinenfehlern	0
15	8 bis 28	Adresse der erweiterten Logouts bei Maschinenfehlern	512

8. Permanente Hauptspeicherzuordnung

Hauptspeicheradresse dez	sed	Länge	Verwendungszweck	Erw. Steuer.
0	00	Doppelwort	PSW für einleitendes Programmladen und neues PSW für Restart	
8	08	Doppelwort	1. CCW für das einl. Programmladen und altes PSW für Restart	
16	10	Doppelwort	2. CCW für das einl. Programmladen	
24	18	Doppelwort	Altes PSW für externe Unterbrechungen	
32	20	Doppelwort	Altes PSW für Supervisoraufrufe	
40	28	Doppelwort	Altes PSW für Programmunterbrechungen	
48	30	Doppelwort	Altes PSW für Maschinenfehler	
56	38	Doppelwort	Altes PSW für EA-Unterbrechungen	
64	40	Doppelwort	Kanalstatuswort(CSW)	
72	48	Wort	Kanaladreßwort (CAW)	
76	4C	Wort	(unbenutzt)	
80	50	Wort	Intervallzeitgeber	
84	54	Wort	(unbenutzt)	
88	58	Doppelwort	Neues PSW für externe Unterbrechungen	
96	60	Doppelwort	Neues PSW für Supervisoraufrufe	
104	68	Doppelwort	Neues PSW für Programmunterbrechungen	
112	70	Doppelwort	Neues PSW für Maschinenfehler	
120	78	Doppelwort	Neues PSW für EA-Unterbrechungen	
128	80	Wort	(unbenutzt)	
132	84	Wort	Externer Unterbrechungsschlüssel auf den Bitstellen 16 bis 31; Bitstellen 0 bis 15 auf Null gesetzt	*
136	88	Wort	SVC-Unterbrechung (Instruktionslänge auf den Bitstellen 13 und 14); Unterbrechungsschlüssel auf den Bitstellen 16 bis 31; übrige Bitstellen auf Null gesetzt)	*
140	8C	Wort	Programmunterbrechung (Instruktionslänge auf den Bitstellen 13 und 14; Unterbrechungsschlüssel auf den Bitstellen 16 bis 31; übrige Bitstellen auf Null gesetzt)	*
144	90	Wort	Adresse bei der Umsetzungsausnahmebedingung (Bitstellen 0 bis 7 auf Null gesetzt)	*
148	94	Wort	Monitorunterbrechung (Klasse auf den Bitstellen 8 bis 15; Bitstellen 0 bis 7 auf Null gesetzt)	
150	96	Halbwort	Unterbrechungsschlüssel bei der Programm-Ereignisregistrierung auf den Bitstellen 0 bis 7 (Bitstellen 8 bis 15 enthalten Nullen)	*
152	98	Wort	Adresse bei der Programm-Ereignisregistrierung (Bitstellen 0 bis 7 auf Null gesetzt)	*
156	9C	Wort	Monitorschlüssel (Bitstellen 0 bis 7 auf Null gesetzt)	
160	A0	Doppelwort	(unbenutzt)	
168	A8	Wort	Kanalidentifikation (Bitstellen 0 bis 3: Typ; Bitstellen 4 bis 15: Modell; Bitstellen 16 bis 31: Länge des erweiterten EA-Logouts)	
172	AC	Wort	Adresse des erweiterten EA-Logouts (Bitstellen 0 bis 7 auf Null gesetzt)	
176	B0	Wort	Begrenzter Logout vom Kanal	
185	B9	Drei Bytes	EA-Adresse bei Unterbrechung (Bitstellen 0 bis 7 auf Null gesetzt)	*
216	D8	Doppelwort	Rettungsgebiet für internen Zeitgeber bei Maschinenfehlern	

Hauptspeicheradresse dez	sed	Länge	Verwendungszweck	Erw. Steuer.
224	E0	Doppelwort	Rettungsgebiet für den Uhrzeitvergleicher bei Maschinenfehlern	
232	E8	Doppelwort	Unterbrechungsschlüssel bei Maschinenfehlern	
248	F8	Wort	Fehlerverursachende Hauptspeicheradresse bei Maschinenfehlern	
252	FC	Wort	Regionsschlüssel (modellabhängig)	
256	100	16 Bytes	Festes Gebiet für Logouts	
352	160	32 Bytes	Rettungsgebiet für die Gleitpunktregister bei Maschinenfehlern	
384	180	64 Bytes	Rettungsgebiet für die allgemeinen Register bei Maschinenfehlern	
448	1C0	64 Bytes	Rettungsgebiet für die Steuerungsregister bei Maschinenfehlern	
512	200	variabel	Gebiet für die erweiterten Logouts bei Maschinenfehlern (Steuerungsregister 15 verweist auf dieses Gebiet; die Länge ist modellabhängig)	

Anmerkung: Ein Sternzeichen in der letzten Spalte besagt, daß der Inhalt nur bei der erweiterten Steuerung signifikant ist.

Anhang E: Übersicht über PL/1

1. Abkürzung von Schlüsselwörtern

Abkürzung	*Schlüsselwort*
*ALLOC	ALLOCATE
*ALLOCN	ALLOCATION
ASM	ASSEMBLER
*ATTN	ATTENTION
AUTO	AUTOMATIC
BIN	BINARY
BUF	BUFFERED
CHAR	CHARACTER
COL	COLUMN
CPLN	COMPLETION
CPLX	COMPLEX
COND	CONDITION
CONN	CONNECTED
CTL	CONTROLLED
CONV	CONVERSION
DEC	DECIMAL
DCL	DECLARE
DFT	DEFAULT
DEF	DEFINED
ENV	ENVIRONMENT
*EXCL	EXCLUSIVE
EXT	EXTERNAL
FOFL	FIXEDOVERFLOW
GOTO	GO TO
INIT	INITIAL
INT	INTERNAL

Abkürzung	*Schlüsselwort*
*IRRED	IRREDUCIBLE
NOCONV	NOCONVERSION
NOFOFL	NOFIXEDOVERFLOW
NOOFL	NOOVERFLOW
NOSTRG	NOSTRINGRANGE
NOSTRZ	NOSTRINGSIZE
NOSUBRG	NOSUBSCRIPTRANGE
NOUFL	NOUNDERFLOW
NOZDIV	NOZERODIVIDE
OFL	OVERFLOW
PIC	PICTURE
PTR	POINTER
POS	POSITION
PREC	PRECISION
PROC	PROCEDURE
*RED	REDUCIBLE
SEQL	SEQUENTIAL
STRG	STRINGRANGE
STRZ	STRINGSIZE
SUBRG	SUBSCRIPTRANGE
UNAL	UNALIGNED
UNBUF	UNBUFFERED
UNDF	UNDEFINEFILE
UFL	UNDERFLOW
VAR	VARYING
ZDIV	ZERODIVIDE

Anmerkung: Die mit einem Sternzeichen versehenen Abkürzungen gelten nur für die im Betriebssystem OS implementierte Sprache

2. PL/1-Statements

Die nachfolgende Statementübersicht ist nur eine begrenzte Auswahl aller in PL/1 zur Verfügung stehenden Statements.

1) marke: PROCEDURE (parameter$_1$, parameter$_2$, ···) [OPTIONS (MAIN)]
[RECURSIVE]
[RETURNS(dateiattribute)] ;

2) BEGIN;

3) END [marke] ;

4) eingangsname: ENTRY [(parameter$_1$, parameter$_2$, ···)]
[RETURNS(dateiattribute)] ;

5) DO;

6) DO WHILE (skalarer ausdruck);

7) DO { pseudovariable / variable } = spezifikation$_1$ [, spezifikation$_2$, ···];
Jede Spezifikation besitzt den folgenden Aufbau:
ausdruck$_1$ [TO ausdruck$_2$ [BY ausdruck$_3$] / BY ausdruck$_3$ [TO ausdruck$_2$]] [WHILE (ausdruck$_4$)]

8) { skalare variable / pseudovariable } [, skalare variable / , pseudovariable] ··· = skalarer ausdruck;

9) { bereichsvariable / pseudobereichsvariable } [, bereichsvariable / , pseudobereichsvariable] ··· = { bereichsausdruck [, BY NAME] / skalarer ausdruck } ;

10) { strukturvariable / pseudostrukturvariable } [, strukturvariable / , pseudostrukturvariable] ··· = strukturausdruck;

11) { strukturvariable / pseudostrukturvariable } [, strukturvariable / , pseudostrukturvariable] ··· = strukturausdruck [, BY NAME];

12) skalare markenvariable [, skalare markenvariable] ··· = { markenkonstante / skalare markenvariable } ;

13) bereichsmarkenvariable [, bereichsmarkenvariable] ··· = { markenkonstante / skalare markenvariable / bereichsmarkenvariable } ;

14) zeigervariable [, zeigervariable] ··· = zeigerausdruck;

15) bereichszeigervariable [, bereichszeigervariable] ··· = { zeigerausdruck / bereichszeigerausdruck } ;

16) IF skalarer ausdruck THEN einheit$_1$
[ELSE einheit$_2$]
Eine Einheit ist hierbei entweder eine einzelne Anweisung, eine DO-Gruppe oder ein BEGIN-Block.

17) { GO TO / GOTO } { markenkonstante / markenvariable } ;

18) CALL eingangsname [(argument$_1$, argument$_2$, ···)];

19) RETURN [(ausdruck)];

20) [marke:];
Leeranweisung

21) STOP;

22) ON bedingung { einheit / SYSTEM; }
Die Einheit ist in der Subsetsprache entweder die Leeranweisung oder eine GOTO-Anweisung; in der vollen Sprache eine beliebige Anweisung außer der RETURN-Anweisung oder ein nicht markierter BEGIN-Block.

23) ON bedingung { [SNAP] einheit / [SNAP] SYSTEM; }

24) SIGNAL bedingung;

25) REVERT bedingung;

26) DECLARE [stufe$_1$] name$_1$ [attribute$_1$] [, [stufe$_2$] name$_2$ [attribute$_2$]] ··· ;

27) OPEN optionsgruppe$_1$ [, optionsgruppe$_2$] ··· ;
Jede Optionsgruppe kann hierbei folgendermaßen aufgebaut sein:

FILE(dateiname) [TITLE(ausdruck)]

[{INPUT / OUTPUT / UPDATE}] [{STREAM / RECORD}] [{DIRECT / SEQUENTIAL}]

[{BUFFERED / UNBUFFERED}] [EXCLUSIVE] [KEYED] [BACKWARDS]

[PRINT] [LINESIZE (ausdruck)] [PAGESIZE (ausdruck)]

28) CLOSE FILE (dateiname$_1$) [, FILE (dateiname$_2$)] ··· ;
In der vollen Sprache kann hinter jeder Option FILE (dateiname$_i$) eine zweite Option, nämlich IDENT (argument) stehen.

29) GET [{FILE (dateiname) / STRING (zeichenkettenvariable)}] [{DATA [(datenliste)] / LIST (datenliste) / EDIT (datenliste) (formatliste)}] [COPY];

30) PUT [{FILE (dateiname) / STRING (zeichenkettenvariable)}] [{DATA (datenliste) / LIST (datenliste) / EDIT (datenliste) (formatliste)}]
[PAGE] [LINE (ausdruck)] [SKIP [(ausdruck)]];

31) FORMAT (formatliste);

32) READ FILE (dateiname) {INTO (variable) / SET (zeigervariable) / IGNORE (ausdruck)} [KEY (ausdruck)]
[KEYTO (zeichenkettenvariable)];

33) WRITE FILE (dateiname) FROM (variable) [KEYFROM (ausdruck)];

34) REWRITE FILE (dateiname) [FROM (variable)] [KEY (ausdruck)];

35) LOCATE variable FILE (dateiname) SET (zeigervariable) [KEYFROM (ausdruck)];

36) DISPLAY (skalarer ausdruck) [REPLY (zeichenvariable)];

3. Attribute in DECLARE-Vereinbarungen

1) Stufennummer: 1 bis 255;
Standardmäßig wird die Stufennummer 1 zugeordnet

2) Dimensionierung: ([untere grenze:] obere grenze
[, [untere grenze:] obere grenze]...)
In der Subsetsprache kann die untere Grenze nicht festgelegt werden; sie ist standardmäßig gleich 1, genau wie in der vollen Sprache, wenn dort die untere Grenze nicht angegeben wird.

3) Arithmetische Datenelemente: [{REAL / COMPLEX}] [{DECIMAL / BINARY}] [{FLOAT / FIXED}] [{(p) / (p[,q])}]
oder
PICTURE'numerische abbildungszeichen'

Die standardmäßig ergänzten Attribute sind unterstrichen worden, d.h. fehlt z.B. die Angabe von DECIMAL oder BINARY, so wird standardmäßig DECIMAL ergänzt. Voraussetzung für die Ergänzung von Attributen ist, daß mindestens ein Attribut angegeben ist. Fehlt die Angabe von Attributen überhaupt, so werden allen mit den Buchstaben I, J, K, L, M, N beginnenden Namen die Attribute BINARY FIXED (15) zugeordnet, d.h. der betreffende Name gilt als Name einer arithmetischen Variablen. Beginnen die Namen mit irgendeinem anderen Buchstaben, dann werden ihnen die Attribute FLOAT DECIMAL (6) zugeordnet.

Bei der Angabe von Genauigkeitsattributen darf man die maximale Genauigkeit nicht überschreiten; wird kein Genauigkeitsattribut bei arithmetischen Variablen angegeben, so erfolgt die Zuordnung einer Standardgenauigkeit:

Andere Attribute	Standardgenauigkeit	Maximalgenauigkeit
FLOAT DECIMAL	(6)	(16)
FLOAT BINARY	(21)	(53)
FIXED DECIMAL	(5, 0)	(15,q)
FIXED BINARY	(15, 0)	(31,q)

Im PICTURE-Attribut dürfen nicht die Abbildungszeichen A und X verwendet werden.

4) Kettendatenelemente: BIT(n) [ALIGNED / UNALIGNED]
oder
CHARACTER(n)
oder
PICTURE'abbildungszeichen'
In der Folge der Abbildungszeichen muß mindestens ein A oder ein X auftreten.

Die maximale Länge für Bitketten bzw. Zeichenketten beträgt 32767.

5) Markenvariablen: LABEL [(markenkonstante$_1$,...)]

6) Zeigervariablen: POINTER

7) Eingangsnamen: ENTRY [(parameter$_1$-attributliste,...)]
RETURNS(attribute)

8) Dateinamen:

```
STREAM | RECORD
INPUT | OUTPUT | UPDATE
SEQUENTIAL | DIRECT
BUFFERED | UNBUFFERED
PRINT
BACKWARDS
KEYED
ENVIRONMENT(optionsliste)
```

9) Attribute, die allen Bezeichnungen (Namen) zugeordnet werden können.

```
INTERNAL | EXTERNAL
STATIC | AUTOMATIC | CONTROLLED | BASED(zeigervariable)
INITIAL(···)
DEFINED
LIKE
POSITION
```

4. Formatelemente

Festkomma: `F(w)` `F(w,d)` `F(w,d,s)`

Gleitkomma: `E(w,d'` `E(w,d,p)`

Komplex: `C(f [,f])`

Abbildung: `P'abbildungszeichenfolge'`

Bitkette: `B[(w)]`

Zeichenkette: `A[(w)]`

R-Formatelement: `R(markenkonstante)` `R(markenvariable)`

Steuerungsformatelemente:

```
X(w)
PAGE
SKIP [(n)]
LINE(n)
COLUMN(n)
```

Beispiele

VAR= -27.443

Formatelement	Nach der Ausgabe
F(4)	b-27
F(8)	bbbbb-27
F(8,Ø)	bbbb-27.
F(8,2)	bb-27.44
F(8,2,1)	b-274.43
E(12,2)	bbb-2.74E+Ø1
E(12,2,3)	bbb-2.74E+Ø1
E(12,2,4)	bb-27.44E+ØØ
E(12,2,6)	-2744.3ØE-Ø2
P'ZZZV.99T'	bb27.44L
P'-----V.999'	b-27.443
P'-----V999'	b-27443
P'9V.99TE+99'	2.74ME+Ø1
P'-9V.999K899'	-2.744+Ø1

Es bedeuten:
- w Feldlänge
- d Anzahl der Dezimalstellen
- s Skalenfaktor
- p Anzahl der signifikaten Ziffern
- f reelles Formatelement

Bei Angabe eines Skalenfaktors s wird der interne Wert vor der Ausgabe mit 10^s multipliziert; bei der Eingabe wird der externe Wert mit 10^s multipliziert, bevor er gespeichert wird.

5. Datenumwandlungen

5.1. Typumwandlungen

1. *Verschlüsselte arithmetische Daten* ⟶ *arithmetische Daten in Zeichenform*
   — Konvertierung in einen Dezimalwert (falls als Dualwert vorliegend)
   — Umwandlung in Zeichenform gemäß der gegebenen Abbildung (PICTURE-Attribut)

2. *Verschlüsselte arithmetische Daten* ⟶ *Zeichenkettendaten*
   a) Gleitpunktdarstellung
      — Anpassung an FLOAT DECIMAL(p), falls notwendig
      — Umwandlung gemäß der Abbildung '-9V.(p-1)9ES99' für den Realteil und 'S9V.(p-1)9ES99' für den Imaginärteil

   b) Festpunktdarstellung
      — Anpassung an FIXED DECIMAL(p,q), falls notwendig
      — Bei $p \geq q \geq 0$ Umwandlung gemäß der Abbildung 'B(p-q)-V9.(q)9' für den Realteil und 'S(p-q)9V9.(q)9' für den Imaginärteil. Ist q=0, so wird der Punkt aus den Abbildungen entfernt und das Abbildungszeichen B linksbündig in die Abbildung eingefügt, so daß die Zahl der Abbildungszeichen gleich bleibt.

      Bei den übrigen Fällen wird die Abbildung '-(p-q)-V(q)-FS(k)9' für den Realteil und die Abbildung 'S(p-q)9V(q)9FS(k)9' für den Imaginärteil zugrunde gelegt. Der Wert von k ist gleich 1, wenn $|q| \leq 9$, gleich 2, wenn $10 \leq |q| \leq 99$ und gleich 3, wenn $100 \leq |q|$ ist.

Anmerkung: Bei komplexen Werten wird ans rechte Ende des Imaginärteils außerdem der Buchstabe I angehängt. Die Länge der sich ergebenden Zeichenkette ergibt sich wie folgt:
a) für reelle Zahlen:
   p+6 bei Gleitpunktdarstellung
   p+3+k bei Festpunktdarstellung (k=0 bei nicht erforderlichem Skalenfaktor)
b) für komplexe Zahlen:
   zwei Mal die Länge des entsprechenden reellen Datenelementes plus 1

Zahlen in Gleitpunktdarstellung werden bei der Umwandlung nur dann gerundet, wenn sie im Rahmen eines PUT-Statements erfolgt.

3. *Verschlüsselte arithmetische Daten* ⟶ *Bitkettendaten*
   — Anpassung an FIXED BINARY(p,0), falls notwendig
   — Entfernung des Vorzeichens
   — Betrachtung des Resultats als BIT(p)

4. *Arithmetische Daten in Zeichenform* ⟶ *verschlüsselte arithmetische Daten*
   Umwandlung in ein verschlüsseltes arithmetisches Datenelement mit dezimaler Basis; Modus, Skalierung und Genauigkeit werden aufgrund der Abbildungszeichen bestimmt.

5. *Arithmetische Daten in Zeichenform* ⟶ *Zeichenkettendaten*
   Das numerische Feld wird einfach als Zeichenkette angesehen (keine Umwandlung).

6. *Arithmetische Daten in Zeichenform* ⟶ *Bitkettendaten*
   — Umwandlung gemäß Pkt. 4.
   — Umwandlung gemäß Pkt. 3.

7. *Zeichenkettendaten* ⟶ *verschlüsselte arithmetische Daten*
   Voraussetzung: Die Zeichenkette muß eine gültige arithmetische Konstante (ohne eingebettete Leerzeichen also) sein; Leerzeichen am Ende und am Anfang der Zeichenkette sind gestattet.

   Die Attribute des Resultats der Umwandlung werden aus der Konstantendarstellung entnommen. Die Umwandlung erfolgt gemäß Pkt. 4.

   Bei Umwandlungen dieser Art in Ausdrücken besitzt jedoch das Ergebnis stets die Attribute DECIMAL FIXED(15, 0).

8. *Zeichenkettendaten* ⟶ *arithmetische Daten in Zeichenform*
   Gleiche Voraussetzung wie bei Pkt. 7.
   Die arithmetische Konstante innerhalb der Zeichenkette wird einfach als arithmetisches Datenelement in Zeichenform angesehen.

9 *Zeichenkettendaten* ⟶ *Bitkettendaten*
   Voraussetzung: Die Zeichenkette darf nur die Zeichen 0 und 1 aufweisen. Bei anderen Zeichen wird die Bedingung CONVERSION gesetzt.

   Jedes der Zeichen 0 bzw. 1 wird in ein Bit 0 bzw. 1 überführt. Die Umwandlung verläuft von links nach rechts; nicht mehr benutzte Zeichen werden nicht geprüft. Die Bitkette wird im Bedarfsfall nach rechts abgeschnitten oder mit 0-Bits aufgefüllt.

10. *Bitkettendaten* ⟶ *verschlüsselte arithmetische Daten*
    Die Bitkette wird so verändert, daß sie die Attribute BIT(56) besitzt. Die Bedingung SIZE tritt auf, wenn dabei 1-Bits verloren gehen. Das Resultat wird mit einem Pluszeichen versehen und als BINARY FIXED(56) betrachtet. Kommt es zu einer solchen Umwandlung in einem Ausdruck, so erfolgt die Anpassung nach BIT(31) mit anschließender Betrachtung als BINARY FIXED(31).

11. *Bitkettendaten* ⟶ *arithmetische Daten in Zeichenform*
    — Umwandlung gemäß Pkt. 10.
    — Umwandlung gemäß Pkt. 4.

12. *Bitkettendaten* ⟶ *Zeichenkettendaten*
    Jedes Bit (0 bzw. 1) wird in das zugeordnete Zeichen (0 bzw. 1) umgewandelt, wobei nur die tatsächlich erforderlichen Bits umgewandelt werden. Die entstandene Zeichenkette wird im Bedarfsfall nach rechts abgeschnitten oder mit Leerzeichen aufgefüllt.

5.2. Arithmetische Anpassung

Die Genauigkeit von umgewandelten arithmetischen Daten in verschlüsselter Form ergibt sich wie folgt:

1. FIXED DECIMAL(p,q) ⟶ FLOAT DECIMAL(p)
2. FIXED BINARY(p,q) ⟶ FLOAT BINARY(p)
3. FIXED BINARY(p,q) ⟶ FLOAT DECIMAL(CEIL(p/3.32))
4. FLOAT BINARY(p) ⟶ FLOAT DECIMAL(CEIL(p/3.32))
5. FIXED DECIMAL(p,q) ⟶ FLOAT BINARY(CEIL(3.32*p))
6. FLOAT DECIMAL(p) ⟶ FLOAT BINARY(CEIL(3.32*p))
7. FIXED BINARY(p,q) ⟶ FIXED DECIMAL(1+CEIL(p/3.32),CEIL(q/3.32))
8. FIXED DECIMAL(p,q) ⟶ FIXED BINARY(1+CEIL(3.32*p),CEIL(3.32*q))

   oder

   ⟶ FIXED BINARY(CEIL(3.32*p),Ø)

   bei Auftreten in einer Umwandlung zur Bitkette

Wertetabelle für die CEIL-Funktion

Y = CEIL(3.32 X)

X	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Y	4	7	10	14	17	20	24	27	30	34	37	40	44	47	50	54	57	60	64	67	70	74	77	80	83	87

X	27	28	29	30	31	32	33
Y	90	93	97	100	103	107	110

Z = CEIL(Y/3.32)

Y	1 bis 3	4 bis 6	7 bis 9	10 bis 13	14 bis 16	17 bis 19	20 bis 23
Z	1	2	3	4	5	6	7

Y	24 bis 26	27 bis 29	30 bis 33	34 bis 36	37 bis 39	40 bis 43	44 bis 46
Z	8	9	10	11	12	13	14

Y	47 bis 49	50 bis 53	54 bis 56	57 bis 59	60 bis 63	64 bis 66	67 bis 69
Z	15	16	17	18	19	20	21

Y	70 bis 73	74 bis 76	77 bis 79	80 bis 82	83 bis 86	87 bis 89	90 bis 92
Z	22	23	24	25	26	27	28

Y	93 bis 96	97 bis 99	100 bis 102	103 bis 106	107 bis 109	110 bis 112	113 bis 116
Z	29	30	31	32	33	34	35

5.3. Datenumwandlungen in Operationen

1. Bitkettenoperationen & und |

   Der erste Operand wird zu BIT(p) umgeformt, der zweite zu BIT(q). Das Resultat besitzt die Charakteristik BIT(MAX(p,q)).

2. Bitkettenoperation ¬

   Der Operand wird zu BIT(p) umgeformt. Das Resultat besitzt die gleiche Charakteristik.

3. Verkettungsoperation ||

   Wenn beide Operanden nicht als Bitketten vorliegen, so werden sie in Zeichenketten umgewandelt. Die Verkettung erfolgt entweder mit Bitketten oder mit Zeichenketten. Besitzt die erste Kette die Länge p_1 und die zweite Kette die Länge p_2, so weist die Ergebniskette die Länge p_1+p_2 auf.

4. Vergleichsoperationen >, ¬>, >=, <, ¬<, <=, =, ¬=

   a) *Algebraischer Vergleich*
   Ist einer der Operanden arithmetisch, entweder in verschlüsselter Form oder in Zeichenform, so wird von beiden die verschlüsselte Form hergestellt.

   b) *Zeichenvergleich*
   Ist einer der Operanden eine Bitkette, der andere eine Zeichenkette, so wird die Bitkette in die Zeichenkette umgewandelt und ein Zeichenvergleich durchgeführt.

   c) *Bitvergleich*
   Sind beide Operanden Bitketten, so findet ein Bitvergleich statt, dem keine Umwandlungen vorausgehen.

   Das Resultat eines Vergleiches besitzt stets die Charakteristik BIT(1).

Anmerkung: Für Programmsteuerungsdaten, daher auch für komplexe arithmetische Daten sind nur die Operatoren = und ¬= erlaubt.

5. Arithmetische Operationen +, -, *, /

Zuerst erfolgt die Umwandlung der Operanden in die verschlüsselte arithmetische Form. Ist dabei ein Operand komplex, so wird auch der andere zu einem komplexen gemacht. Bei unterschiedlicher Darstellungsart wird für beide die Gleitpunktdarstellung hergestellt. Bei unterschiedlichen Basen kommt die duale Zahlenbasis (BINARY) zum Zuge. Entsprechend dieser Regeln finden die notwendigen Umwandlungen bzw. Anpassungen statt.

Besitzt der erste Operand die Charakteristik FLOAT(p_1), der zweite die Charakteristik FLOAT(p_2), dann weist das Resultat die Eigenschaft FLOAT(MAX(p_1, p_2)) auf.

Besitzt der erste Operand die Charakteristik FIXED(p_1, q_1), der zweite die Charakteristik FIXED(p_2, q_2), so ergeben sich folgende Charakteristiken FIXED(p, q) für das Resultat

Operation	p	q
+ -	$p=1+q+MAX(p_1-q_1, p_2-q_2)$	$q=MAX(q_1, q_2)$
*	p_1+p_2+1	$q=q_1+q_2$
/	p=15 (bei DECIMAL) p=31 (bei BINARY)	$q=p-p_1+q_1-q_2$

6. Potenzierung **

Für beide Operanden wird, falls notwendig, zuerst die verschlüsselte arithmetische Form hergestellt.

a) *Fall 1*
Die Basis besitzt die Charakteristik FIXED(p_1, q_1), der Exponent n ist eine ganzzahlige Konstante ohne Vorzeichen.

Das Resultat wird durch fortlaufende Multiplikationen gefunden und besitzt die Eigenschaft FIXED(p, q), wobei $p=n*p_1-n-1$ und $q=n \cdot q_1$ ist.

b) *Fall 2*
Besitzt die Basis der Potenz die Charakteristik FLOAT(p_1) und der Exponent die Charakteristik FIXED(p_2, 0), so ergibt sich eine Potenz mit FLOAT(p_1), deren Zahlenbasis (DECIMAL oder BINARY) die gleiche ist wie die der Potenzbasis.

c) *Fall 3*
Zum Fall 3 sind alle anderen denkbaren Möglichkeiten zusammengefaßt. Beide Operanden werden in die Gleitpunktdarstellung umgewandelt. Sind beide Operanden nicht dezimaler Natur (DECIMAL), so werden beide in Dualzahlen umgewandelt. Das Resultat besitzt die Eigenschaft FLOAT(p), wobei $p=MAX(p_1, p_2)$ ist. Die Zahlenbasis des Resultats ist die duale (Attribut BINARY), wenn nicht beide Operanden die dezimale Basis aufweisen.

6. Eingefügte Funktionen

Funktionsname	Attribute der Argumente	Definition der Funktion	Attribute des Funktionswertes oder Bemerkungen
ABS(x)	REAL	$\|x\|$	Gleiche Attribute wie das Argument
* ABS(x)	COMPLEX(p, q)	$\sqrt{a^2+b^2}$ für x=a+bi	REAL (p+1, q)
MIN(x_i, ...) MAX(x_i, ...)	REAL REAL	Kleinster x-Wert Größter x-Wert	Genauigkeit gleich der größten Genauigkeit aller Argumente
MOD(x, y)	REAL	Positiver Rest nach Division von x durch y, wobei der Quotient ganzzahlig ist	Rest reell
SIGN(x)	REAL FLOAT	+ 1 für x > 0 0 für x = 0 - 1 für x < 1	REAL FIXED BINARY
FIXED(x, p, q)	FLOAT	x in Festkommadarstellung	Genauigkeit (p, q); fehlen p oder q → standardmäßige Genauigkeit
FLOAT(x, p)	FIXED	x in Gleitkommadarstellung	Genauigkeit (p); fehlt p Standardgenauigkeit
FLOOR(x) CEIL(x) TRUNC(x)	REAL(p, q) REAL(p, q) REAL(p, q)	Größte ganze Zahl, die ≦ x ist kleinste ganze Zahl, die > x ist Bei x ≦ 0 FLOOR(x) bei < 0 CEIL(x)	REAL (p-q+1, 0)
BINARY(x, p[, q]) DECIMAL(x, p[, q]) PRECISION(x, p[, q])	nicht binär nicht dezimal beliebig	x mit Attributen BIN(p, q) x mit Attributen DEC(p, q) x mit Genauigkeit(p, q)	Besitzt x das Attribut FLOAT, so ist q wegzulassen
ADD(x, y, p[, q]) MULTIPLY(x, y, p[, q]) DIVIDE(x, y, p[, q])	numerische Daten	x+y mit Genauigkeit (p, q) x·y mit Genauigkeit (p, q) x/y mit Genauigkeit (p, q)	Besitzt x oder bzw. und y das Attribut FLOAT, so ist q wegzulassen
* COMPLEX(a, b)	REAL	x=a+bi	COMPLEX
* REAL(x) * IMAG(x)	COMPLEX COMPLEX	a bei x = a + bi a bei x = a + bi	REAL
* CONJG(x)	COMPLEX	a - bi bei x = a + bi	COMPLEX
EXP(x)	FLOAT	exp(x)	FLOAT
LOG(x) LOG10(x) LOG2(x)	FLOAT REAL FLOAT REAL FLOAT	ln(x) lg(x) $^2\log(x)$	FLOAT; Fehler bei x ≦ 0
ATAND(x) ATAN(x) TAND(x) TAN(x) SIND(x) SIN(x) COSD(x) COS(x) TANH(x) SINH(x) COSH(x)	REAL FLOAT FLOAT REAL FLOAT REAL FLOAT REAL FLOAT FLOAT REAL FLOAT FLOAT REAL FLOAT FLOAT FLOAT	arctan (x) im Gradmaß arctan (x) im Bogenmaß tan(x^o) tan(x) sin(x^o) sin(x) cos(x^o) cos(x) tanh(x) sinh(x) cosh(x)	FLOAT
ATANH(x)	FLOAT	ar tanh(x)	FLOAT; Fehler bei $\|x\| \geqq 1$
ATAN(y, x)	REAL FLOAT	siehe unter 1)	FLOAT; Fehler bei x=y=0; Bogenmaß
ATAND(y, x)	REAL FLOAT	wie ATAN	FLOAT; Fehler bei x=y=0; Gradmaß
SQRT(x)	FLOAT	$\sqrt{x}$	FLOAT; Fehler bei x<0
ERF(x) ERFC(x)	REAL FLOAT REAL FLOAT	$\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dx$ 1 - ERF(x)	FLOAT
BIT(x[, p]) CHAR(x[, p])	Kette	x als Bitkette x als Zeichenkette	BIT(p); fehlt p, so wird die Länge von x genommen CHAR(p); fehlt p, so wird die Länge von x genommen

Funktionsname	Attribute der Argumente	Definition der Funktion	Attribute des Funktionswertes oder Bemerkungen
SUBSTR(x, i, j)	x Kette der Länge k; zweites und drittes Argument Ausdrücke, die nach Umwandlung in ganze Zahlen i und j ergeben	Teilkette, beginnend an der i-ten Stelle von x und j Zeichen bzw. Bits umfassend. Ist i>k, so ergibt sich die Nullkette.	
INDEX(x, y)	Ketten	Stelle von x, von der aus y Teilkette von x ist	FIXED BINARY (15)
HIGH(d) LOW(d)	FIXED DEC(p, 0) FIXED DEC(p, 0)	Jedes Byte Des Funktionswertes enthält sed FF Jedes Byte des Funktionswertes enthält sed 00	} CHARACTER(d)
REPEAT(x, n)	x Kette n ganze Dezimalzahl	x wird mit sich selbst n-mal verkettet	
UNSPEC(x)	x skalare Variable	Bitkette, die gleich der verschlüsselten internen Darstellung des Argumentes ist	
BOOL(x, y, w)	x, y, w Bitketten; w wird in BIT(4) umgewandelt. Sei w='$n_1 n_2 n_3 n_4$'B, dann gilt	Als Funktionswert ergibt sich die Bitkette z, deren Bits gemäß der nebenstehenden Tabelle gebildet werden	x_j | y_j | z_j 0 | 0 | n_1 0 | 1 | n_2 1 | 0 | n_3 1 | 1 | n_4 Besitzen x und y unterschiedliche Längen, so wird die kürzere Kette nach rechts mit Nullen aufgefüllt; der Funktionswert z besitzt die größere Länge
SUM(x)	} Bereich	Summe aller Bereichselemente	} FLOAT
PROD(x)		Produkt aller Bereichselement	
ALL(x)		Jedes Element wird in eine Bitkette umgewandelt. Die einzelnen Bits der Bitketten werden logisch verknüpft	UND-Verknüpfung ODER-Verknüpfung } Der Funktionswert ist eine Bitkette
ANY(x)			
* POLY(a, x)	a(m:n) sind x(p:q) Vektoren	$a(m)+\sum_{j=1}^{n-m} a(m+j)\prod_{i=0}^{j-1} x(p+i)$	Für m=n ist der Funktionswert gleich a_m
* ONFILE	—	Name der Datei, für die die letzte EA-Operation ausgeführt wurde	
* ONLOC	—	Name der Prozedur, in der die letzte Unterbrechung festgestellt wurde	
* ONSOURCE	—	Inhalt des Feldes, das gerade verarbeitet wurde, als die letzte CONVERSION-Bedingung auftrat	
* ONCHAR	—	Zeichen, das zum Auftreten der letzten CONVERSION-Bedingung führte	
* ONKEY	—	Schlüssel des Satzes, der bei der Übertragung eine Unterbrechung verursachte	
* ONCODE	—	Der Funktionswert mit den Attributen BINARY FICED(15) gibt den Fehlercode der letzten Unterbrechung an	
* DATAFIELD	—	Inhalt des Datenfeldes, das zur Unterbrechungsbedingung NAME führte	
ADDR(x)	beliebige Variable	Zeigerwert, der zur Identifizierung von x dient (Adresse)	
NULL	—	Nullzeiger; identifiziert kein Datenelement	
DATE	—	JJMMTT (Datum)	Zeichenkette (Jahr, Monat, Tag)
TIME		hhmmssttt (Zeit)	Zeichenkette (Stunden, Minuten, Sekunden, Millisekunden)
* LINENO(dateiname)	FILE PRINT	Lfd. Zeile der Druckdatei	BINARY FIXED(15)
* COUNT(dateiname)	FILE	Anzahl der skalaren Elemente, die bei der letzten GET/PUT-Anweisung übertragen wurden (BINARY FIXED(15))	
ROUND(ausdruck, d)	d ganze Dezimalzahl	Der Ausdruck wird auf die d-te Stelle rechts vom Dezimalpunkt gerundet, ist d<0, so erfolgt die Rundung auf die d-te Stelle links vom Dezimalpunkt. Ist der Ausdruck binär, so gibt d die Anzahl der Bitstellen an	
* STRING(strukturname)	Stufennummer	Zeichenkette, die aus der Verkettung aller Strukturelemente entsteht.	

Anmerkung: 1) $\arctan \frac{y}{x}$ für x<0

$\frac{\pi}{2}$ für x=0 und y>0

$-\frac{\pi}{2}$ für x=0 und y<0

$\pi + \arctan \frac{y}{x}$ für x<0 und $y \geq 0$

$-\pi + \arctan \frac{y}{x}$ für x<0 und y<0

7. Pseudovariablen

Es gibt folgende Pseudovariablen

COMPLEX	REAL
IMAG	STRING
ONCHAR	SUBSTR
ONSOURCE	UNSPEC
PRIORITY	

Da die Pseudovariablen den namensgleichen eingefügten Funktionen (siehe Abschnitt 6 dieses Anhangs) entsprechen, genügt für sie eine kurze Beschreibung.

1) COMPLEX(x_1,x_2)

Durch diese Pseudovariable wird der Realteil eines komplexen Wertes der Variablen x_1, der Imaginärteil des gleichen Wertes der Variablen x_2 zugewiesen. Nur ein komplexer Wert kann dieser Pseudovariablen zugewiesen werden. Als Laufvariable in einer DO-Anweisung darf diese Pseudovariable nicht benutzt werden.

2) IMAG(x)

Mit Hilfe dieser Pseudovariablen kann der Imaginärteil eines komplexen Wertes dem Imaginärteil einer komplexen Variablen x zugewiesen werden.

3) ONCHAR

Durch diese Pseudovariable wird der augenblickliche Wert der eingefügten Funktion ONCHAR ersetzt. Der ihr zugewiesene Wert wird in eine Zeichenkette der Länge 1 umgewandelt. Der neue Wert wird dann beim erneuten Versuch, die Umwandlung zu vollziehen, die zum Auftreten der CONVERSION-Bedingung führte, verwendet.

4) ONSOURCE

Durch diese Pseudovariable wird der augenblickliche Wert der eingefügten Funktion ONSOURCE ersetzt. Der Wert, der der Pseudovariablen zugewiesen werden soll, wird vor der Zuweisung in eine Zeichenkette umgewandelt, die erforderlichenfalls nach rechts mit Leerzeichen aufgefüllt wird. Ziel dieser Umwandlung ist die Anpassung des Wertes an das Feld, das bei einer Umwandlung das Auftreten der CONVERSION-Bedingung veranlaßte. Die neue Zeichenkette wird benutzt, wenn der Umwandlungsversuch wiederholt wird.

5) PRIORITY(x)

Durch Wertzuweisung zu dieser Pseudovariablen wird die Priorität der Task, die durch die Taskvariable x gekennzeichnet ist, geändert. Der zuzuweisende Wert muß die Attribute BIN FIXED(15) besitzen, notfalls erfolgt vor der Zuweisung eine Umwandlung. Die Priorität wird relativ zu der Priorität angesehen, die die unmittelbar vor der Zuweisung ablaufende Task aufweist. Die Taskvariable x kann dabei eine aktive oder inaktive Task einschließlich der gerade ablaufenden kennzeichnen. Bei Weglassung von x gilt die Taskvariable als angesprochen, die die gegenwärtig ablaufende Task repräsentiert. Als Laufvariable in einer DO-Anweisung darf diese Pseudovariable nicht verwendet werden.

6) REAL(x)

Mit Hilfe dieser Pseudovariablen kann der Realteil eines komplexen Wertes oder ein reeller Wert dem Realteil einer komplexen Variablen x zugewiesen werden.

7) STRING(x)

Durch diese Pseudovariable wird erreicht, daß alle Elemente einer Datengruppierung 1) namens x (Bereich oder Struktur) nacheinander, d.h. in der Reihenfolge ihrer

---

1) Auch als Datenaggregat bezeichnet.

Anordnung, mit Teilen der zugewiesenen Zeichenkette gefüllt werden (Stück bei Stück). Dieses sequentielle Zuweisen der Teilketten hört in dem Augenblick auf, in dem entweder alle Elemente von x Wertzuweisungen erfahren haben oder kein Teilstück der Kette mehr vorhanden ist, die Ursprung der Zuweisung ist, die also auf der rechten Seite des Ergibtzeichens in einer Ergibtanweisung steht. Im letzteren Fall werden die verbleibenden Elemente von x mit Leerzeichen gefüllt oder aber auf die Länge Ø gebracht, wenn es sich um Zeichenketten variabler Länge (Attribut VARYING) handelt. Im übrigen werden die Elemente von x, die Zeichenketten variabler Länge darstellen, bis zu ihrer Maximallänge aufgefüllt.

In folgenden Fällen kann diese Pseudovariable nicht benutzt werden:

a) als Laufvariable in einer DO-Anweisung,
b) in der KEYTO-Option einer READ-Anweisung,
c) in der INTO-Option einer READ-Anweisung,
d) in Datenlisten von GET-Anweisungen

8) SUBSTR(x_1, x_2 , x_3)

Durch diese Pseudovariable wird erreicht, daß eine Kette einer Teilkette von x_1 zugewiesen wird, die bei x_2 beginnt und x_3 Stellen lang ist. Fehlt x_3, so erfolgt die Zuweisung bis zum Ende von x_1. Der nicht herausgegriffene Teil der Kette x_1 bleibt unverändert.

9) UNSPEC(x)

Durch diese Pseudovariable erfolgt die direkte Zuweisung einer Bitkette zur Variablen x; dabei erfolgt grundsätzlich keine Datenumwandlung. Die zugewiesene Bitkette wird jedoch im Bedarfsfall nach rechts mit Nullen so aufgefüllt, daß alle Bitstellen von x abgedeckt werden können. Wenn x eine Kette variabler Länge darstellt, so wird in die Zuweisung auch das Längenfeld mit einbezogen.

8. ON-Bedingungen

a) Eingabe/Ausgabe-Bedingungen

	Bedingung	Auftreten durch	Standardsystem-Maßnahme
	ENDFILE(dateiname)	Leseversuch nach dem letzten Satz	Kommentar und Setzen der ERROR-Bedingung
	ENDPAGE(dateiname)	Drucken, obwohl die letzte Zeile einer Seite bereits gedruckt ist	Vorschub auf eine neue Seite
	TRANSMIT(dateiname)	andauernder EA-Fehler	Kommentar und Setzen der ERROR-Bedingung
S	UNDEFINEDFILE(dateiname)	Versuch, eine Datei zu eröffnen, die nicht verfügbar ist	Kommentar und Setzen der ERROR-Bedingung
S	NAME(dateiname)	Eine DATA-gesteuerte GET-Anweisung bezieht sich auf einen nicht vorhandenen Variablennamen	Kommentar; das fehlerhafte Feld bleibt unberücksichtigt
	KEY(dateiname)	Bei einer READ- oder REWRITE-Anweisung kann der Schlüssel nicht gefunden werden; bei einer WRITE- oder LOCATE-Anweisung existiert der Schlüssel bereits	Kommentar und Setzen der ERROR-Bedingung
	RECORD(dateiname)	Die Satzlänge eines in einer READ- oder REWRITE-Anweisung angesprochenen Satzes stimmt nicht mit dem Umfang der Strukturvariablen überein	Kommentar und Setzen der ERROR-Bedingung

b) Programmprüfungsbedingungen

	Bedingung	Auftreten durch	Standardsystem-Maßnahme
▫	SUBSCRIPTRANGE	Ein Index liegt außerhalb der für ihn definierten Grenzen. Das Resultat ist unbestimmt	Kommentar und Setzen der ERROR-Bedingung
▫	CHECK(liste von bezeichnungen)	Als Bezeichnungen dürfen die Namen von nichtindizierten skalaren Variablen, Bereichsvariablen sowie Markenkonstanten und die Namen von Eingangsstellen gewählt werden. Die Bedingung wird immer dann gesetzt, wenn das markierte Statement aufgeführt wird oder wenn sich der Wert der in der Liste aufgeführten Variablen ändert.	Es werden die Bezeichnungen und (bei Variablen) ihre neuen Werte herausgeschrieben
▫	STRINGRANGE	Die Längen der Argumente beim Aufruf der eingefügten Funktion SUBSTR erfüllen nicht die Voraussetzungen, die für den Aufruf von SUBSTR gelten, d.h. die zu bildende Teilkette überschreitet die Begrenzungen der Kette.	Der Programmablauf wird fortgesetzt. Als Resultat des Aufrufs von SUBSTR wird jedoch eine Teilkette gebildet, die innerhalb der Begrenzungen der Kette liegt.

c) Bedingungen, die vom Programmierer eingeführt werden

	Bedingung	Auftreten durch	Standardsystem-Maßnahme
	CONDITION(bezeichnung)	SIGNAL-Anweisung	Kommentar und Fortführung des Programmes

d) Berechnungsbedingungen

	Bedingung	Auftreten durch	Standard-Maßnahme
●	CONVERSION	Bei einer Umwandlung von Zeichenkettendaten in einen anderen Datentyp liegen ungültige Daten vor. Das Resultat ist unbestimmt.	Kommentar und Setzen der ERROR-Bedingung
●	FIXEDOVERFLOW	Bei der Verarbeitung von arithmetischen Daten in Festkommadarstellung wird die maximal mögliche Feldlänge überschritten. Das Resultat wird nach links abgeschnitten.	Kommentar und Setzen der ERROR-Bedingung

	Bedingung	Auftreten durch	Standardsystem-Maßnahme
●	OVERFLOW	Der Exponent einer Gleitkommazahl ist größer als das erlaubte Maximum	Kommentar und Setzen der ERROR-Bedingung
▫	SIZE	Der bei einer Eingabeoperation zu speichernde Wert oder das Ergebnis einer Ergibtanweisung überschreitet die Länge des empfangenden Feldes. Das Resultat ist unbestimmt.	Kommentar und Setzen der ERROR-Bedingung
●	UNDERFLOW	Der Exponent einer Gleitkommazahl ist kleiner als das erlaubte Minimum. Die Gleitkommazahl wird durch 0 ersetzt.	Kommentar und Fortführung der Berechnung
●	ZERODIVIDE	Division durch Null; die Bedingung tritt sowohl bei einer Festkommadivision als auch bei einer Gleitkommadivision auf. Das Ergebnis ist unbestimmt.	Kommentar und Setzen der ERROR-Bedingung

e) Bedingungen für Systemmaßnahmen

	Bedingung	Auftreten durch	Standardsystem-Maßnahme
	ERROR	Das Programm muß auf Grund eines Fehlers anormal beendet werden.	Setzen der FINISH-Bedingung. Siehe Anm. 5)
	FINISH	Beendigung des Programmes	Rückkehr zum Supervisor

Anmerkungen:

1) Die durch ein ● gekennzeichneten Bedingungen sind zu Programmanfang im wirksamen Status; sie können durch Bedingungspräfixe in den unwirksamen Status versetzt werden.

2) Die durch ein ▫ gekennzeichneten Bedingungen sind zu Programmanfang im unwirksamen Status; sie können durch Bedingungspräfixe in den wirksamen Status versetzt werden.

3) Alle nicht gekennzeichneten Bedingungen befinden sich ständig im wirksamen Status.

Anhang F: Übersicht über die FORTRAN-Sprache

1. Ausdrücke

1) Regeln für die Berechnung von Ausdrücken mit gemischtem Modus

a) Potenzen mit ganzzahligen Exponenten werden durch wiederholte Multiplikation berechnet. Potenzen mit reellen Exponenten werden über den natürlichen Logarithmus berechnet. Komplexe Exponenten sind nicht erlaubt.

b) Bei gemischtem Modus werden ganzzahlige Größen in reelle Größen und reelle Größen in komplexe Größen umgewandelt. Bei unterschiedlicher Genauigkeit (Länge) der Operanden erfolgt stets eine Umwandlung in Richtung der höheren Genauigkeit; reelle Größen einfacher Genauigkeit werden also stets in reelle Größen erhöhter (doppelter) Genauigkeit umgewandelt.

2) Regeln für aufeinanderfolgende Operationen

a) Unter keinen Umständen dürfen zwei beliebige Operatoren aufeinander folgen.

b) Der logische Operator .NOT. darf jedoch einem anderen logischen Operator folgen, doch in keinem anderen Fall dürfen zwei logische Operatoren aneinander stoßen.

3) Klammerregeln

a) Klammern können in Ausdrücken so verwendet werden, wie es in der Mathematik üblich ist, d.h. zur Änderung der Reihenfolge von Operationen.

b) Durch Klammern können nur Ausdrücke gruppiert werden; sie bedeuten jedoch keine Multiplikation.

c) Nicht benötigte Klammern beeinflussen den Rechengang nicht.

4) Rangfolge der Operationen

a) Wenn die Reihenfolge, in der die Operationen eines Ausdruckes ausgeführt werden sollen, nicht explizit durch gesetzte Klammern festgelegt ist, so werden die Operationen in der nachstehenden Reihenfolge ausgeführt:

I. Berechnung von Funktionswerten
II. Potenzierungen
III. Multiplikationen und/oder Divisionen
IV. Additionen und/oder Subtraktionen
V. Vergleichsoperationen: .LE. , .LT.,
.EQ. , .NE. , .GT. , .GE.
VI. Logische Operation .NOT.
VII. Logische Operation .AND.
VIII. Logische Operation .OR.

b) Treten in einem Ausdruck mehrere Operationen der gleichen Rangstufe auf, so werden diese von links nach rechts ausgeführt, sofern nicht Klammernpaare eine andere Reihenfolge bestimmen. Eine Ausnahme bilden jedoch die Potenzierungen, diese werden von rechts nach links ausgeführt.

2. Datenlisten

1) Konstanten dürfen in Listen nicht aufgenommen werden.
2) Die einzelnen Elemente einer Liste müssen durch Kommas voneinander getrennt werden.
3) In Listen ist eine Schleifenindizierung erlaubt (nach dem Typ der DO-Anweisung).
4) Ineinandergeschachtelte Schleifenindizierungen sind bei Listen ebenfalls möglich. Jede Indizierungsstufe muß dabei in Klammern eingeschlossen werden und durch Kommas von der nachfolgenden getrennt werden.
5) Ein gesamter Bereich kann dadurch angesprochen werden, daß man sich innerhalb einer Datenliste nur auf seinen Namen bezieht.

3. Indizes

Indizes müssen in der Form v
k
v±k
k*v
k*v±c angegeben werden.

Hierbei bedeutet v eine Variable des ganzzahligen Typs, k und c sind ganzzahlige Konstanten. Nach Umwandlung des Indexausdruckes in einen ganzzahligen Wert muß ein positiver Wert vorliegen, der den Maximalwert des betreffenden Index nicht überschreiten darf.

4. Formularsteuerung beim Drucker

Das erste Zeichen der zugehörigen FORMAT-Vereinbarung wird als Steuerzeichen für den Druckermechanismus interpretiert. Es gibt folgende Steuerzeichen:

b Einfacher Zeilentransport vor dem Drucken der nächsten Zeile
0 Doppelter Zeilentransport vor dem Drucken der nächsten Zeile
+ Kein Zeilentransport vor dem Drucken der nächsten Zeile
1 Vorschub auf den Anfang einer neuen Seite vor dem Drucken (der Beginn einer neuen Seite muß durch eine Lochunge im Kanal 1 des Lochbandes markiert sein).

Anmerkung: b bedeutet hier das Leerzeichen

5. Formatangaben

Angabe	Interne Form	Externe Form	Beispiele: Angabe	Beispiele: Externe Form
Iw	ganzzahlig	ganzzahlig	I7	bb-4567
Ew.d	reell*4	Gleitpunktdarstellung	E9.2	-0.45E+02
Fw.d	reell	Festpunktdarstellung	F9.3	bb-45.789
Dw.d	reell*8	Gleitpunktdarstellung	D9.2	-0.45D+02
wH	alphamerisch	alphamerisch	3HEXP	EXP
wX	Leerzeichen	Leerzeichen	3X	bbb
Aw	reell oder ganzzahlig	alphamerisch	A3	EXP
* Gw	ganzzahlig	ganzzahlig	G7	bb-4567
* Gw.d	reell	Gleitpunkt- oder Festpunktdarstellung	G9.3	bb-45.789
* Lw	logisch	T oder F	L3	bbF

Anmerkungen: a) Die mit einem Sternzeichen versehenen Zeilen gelten nur für die volle Sprache.
b) Komplexe Werte werden durch ein Paar von solchen Formatangaben spezifiziert, die für reelle Werte gelten.
c) b bedeutet hier das Leerzeichen.

6. FORTRAN-Statements

1) Ergibtanweisungen

a) Arithmetische Ergibtanweisung

a = b

Hierbei bedeutet a eine ganzzahlige, reelle oder komplexe Variable; b ist ein arithmetischer Ausdruck. Komplexe Variable bzw. komplexe Ausdrücke dürfen nur in der vollen Sprache benutzt werden.

b) Logische Ergibtanweisung

a = b

Hierbei bedeutet a eine logische Variable und b ist ein logischer Ausdruck. Diese Ergibtanweisungen dürfen nur in der vollen Sprache gebildet werden.

2) Steueranweisungen bzw. -vereinbarungen

a) GO TO n
Hierbei ist n die Statementnummer einer Anweisung. Es erfolgt ein Sprung zur Anweisung mit der Nummer n.

b) GO TO $(n_1, n_2, \ldots, n_m)$, i
Hierbei bedeutet i eine nichtindizierte ganzzahlige Variable; die n_j mit $1 \leq j \leq m$ sind Statementnummern von Anweisungen. Bei $1 \leq i \leq m$ erfolgt ein Sprung zur Anweisung mit der Nummer n_i. Im anderen Fall wird das nächstfolgende Statement ausgeführt.

c) ASSIGN n TO i *)
Hierbei bedeutet n eine Statementnummer und i ist eine ganzzahlige Variable, die nur in ASSIGN-Anweisungen und in Anweisungen für den gesetzten Sprung verwendet werden darf. Der Variablen i wird der Wert n zugewiesen.

d) GO TO i, $(n_1, n_2, \cdots, n_m)$ *)
Unter i ist eine ganzzahlige Variable zu verstehen, deren Wert aus einer ASSIGN-Anweisung hervorgegangen ist; unter den n_j sind Statementnummern zu verstehen. Der Wert von i muß gleich n_1, n_2, $\cdots$, n_m sein. Es erfolgt eine Verzweigung zur Anweisung mit der Nummer, die die Variable i als Wert angenommen hat.

e) IF (a) n_1, n_2, n_3
Hierbei bedeutet a einen arithmetischen Ausdruck, n_1, n_2, n_3 sind die Statementnummern von Anweisungen. Es erfolgt eine Verzweigung zum Statement mit der Nummer n_1, wenn $a<0$ ist, zum Statement mit der Nummer n_2, wenn $a=0$ ist und zum Statement mit der Nummer n_3, wenn $a>0$ ist.

f) IF (a) s
Hierbei stellt a einen logischen Ausdruck dar und s ist irgendein Statement mit Ausnahme von Vereinbarungen, DO-Anweisungen oder anderen logischen IF-Anweisungen. Besitzt a den Wert .TRUE. , so wird die Anweisung s ausgeführt, im anderen Fall wird s nicht ausgeführt.

g) DO n $i=m_1, m_2, m_3$
DO n $i=m_1, m_2$

Hierbei ist n die Statementnummer eines Statements, das jedoch keine GO TO-Anweisung, keine arithmetische IF-Anweisung und keine andere DO-Anweisung sein darf; außerdem darf diese Anweisung erst nach der DO-Anweisung auftreten. Unter i ist eine ganzzahlige Variable zu verstehen; m_1, m_2, m_3 sind ganzzahlige Konstanten ohne Vorzeichen oder nichtindizierte ganzzahlige Variablen, denen ein positiver Wert zugewiesen sein muß. Wenn m_3 nicht angegeben ist, so wird es zu 1 angenommen. Die der DO-Anweisung folgenden Anweisungen bis einschließlich der Anweisung, die durch die Statementnummer n markiert ist, werden zyklisch durchlaufen, beginnend mit i=1. Für jeden weiteren Durchlauf wird der Wert von i um m_3 erhöht. Der letzte Schleifendurchlauf erfolgt mit dem maximalen Wert von i, der m_2 nicht überschreitet. Nach dem Verlassen der Schleife wird die Anweisung ausgeführt, die dem Statement mit der Nummer n unmittelbar folgt.

h) CONTINUE
Diese Vereinbarung dient dazu, die Einschränkung zu umgehen, die besagt, daß die letzte Anweisung der Schleifenspanne einer DO-Anweisung keine der nachfolgenden Anweisungen sein darf: GO TO, IF (arithmetisch).

i) PAUSE
PAUSE n
PAUSE 'nachricht' *)
Hierbei bedeutet n eine fünfstellige ganze Zahl ohne Vorzeichen. Wird weder n noch 'nachricht' codiert, so wird für n der Wert 00000 angenommen. Das Statement wird einschließlich der fünfstelligen ganzen Zahl n oder der Nachricht ausgeschrieben. Der Computer unterbricht außerdem seine automatische Arbeitsweise und erwartet einen Bedienereingriff, bevor er die Verarbeitung fortsetzt.

Im Betriebssystem OS muß der Bediener

```
REPLY yy,'z'
```

eintasten, bevor die Verarbeitung wieder aufgenommen wird; hierbei bedeutet yy die zweiziffrige Nachrichtennummer, die als Teil der PAUSE-Nachricht vom Computer ausgegeben wurde, während z ein beliebiges Zeichen ist.

Im Betriebssystem DOS braucht der Maschinenbediener nur die Endtaste am Konsol zu drücken.

Nach Eingehen der Antwort geht die Verarbeitung weiter.

j) STOP n
STOP
Hierbei bedeutet n eine fünfstellige ganze Zahl ohne Vorzeichen; wird n nicht niedergeschrieben, so wird es zu 00000 angenommen. Es wird bei n ≠ 0 die Nachricht

```
STOP n
```

anderenfalls nur

```
STOP
```

ausgegeben; anschließend wird die Programmablaufsteuerung an das Betriebssystem zurückgegeben.

k) END
Diese Vereinbarung zeigt das Ende des Quellenprogrammes an.

Anmerkung: Die mit einem Sternzeichen versehenen Statements haben nur für die volle Sprache Gültigkeit.

3) Eingabe/Ausgabe-Statements

a) READ (a, b) liste
READ (a, b, END=c, ERR=d) liste *)
WRITE (a, b) liste

Durch a ist die Nummer einer Datei, d.h. letzten Endes einer Eingabe/Ausgabe-Einheit, festzulegen; b ist die Statementnummer der zugehörigen FORMAT-Vereinbarung. Durch c bzw. d werden die Nummern der Anweisungen festgehalten, die beim Auftreten der Bedingungen „Dateiende" bzw. „EA-Fehler" angesprungen werden sollen.

b) READ (a) liste
WRITE (a) liste
Hierbei bedeutet a wiederum die Referenznummer der Datei. Eingabe und Ausgabe erfolgen in der Form, wie die Daten im Hauptspeicher verschlüsselt sind.

c) READ (a, x) *)
WRITE (a, x) *)
Hierbei bedeutet a die Nummer einer Datei (Einheitennummer) und x ist der Name einer Namensliste (NAMELIST).

d) DEFINE FILE $a_1(m_1, r_1, f_1, v_1), \cdots$
Mit einem einzigen Statement können soviele Dateien definiert werden, wie es erforderlich ist. Hierbei bedeuten die a_j die Referenznummern der Dateien. Durch m_j ist die maximale Anzahl der in der betreffenden Datei enthaltenen Sätze anzugeben, während mit r_j die maximale Satzlänge festzuhalten ist. Durch v_j wird die der Datei beigeordnete Variable spezifiziert. Nach Abschluß jeder READ-, WRITE- oder FIND-Anweisung wird der Variablen v_j der Wert zugewiesen, der gleich der Satznummer des Satzes ist, der auf den durch die genannten Anweisungen angesprochen Satz folgt.

Durch f_j=L wird angezeigt, daß die Sätze entweder mit Format oder formatfrei gelesen bzw. geschrieben werden, f_j=E bedeutet, daß die Form der ein- bzw. auszugebenden Sätze durch eine FORMAT-Vereinbarung gesteuert wird; f_j=U heißt schließlich, daß die von den E/A-Operationen betroffenen Sätze formatfrei gelesen bzw. geschrieben werden.

e) FIND (a'r)
Hierbei bedeutet a die Nummer der Datei und r spezifiziert die Nummer des Satzes innerhalb der Datei, der aufgesucht werden soll.

f) READ (a'r, b) liste
READ (a'r) liste
WRITE (a'r, b) liste
WRITE (a'r) liste

Durch diese Anweisungen kann direkt zu den Sätzen der durch die Nummer a spezifizierten Datei zugegriffen werden; der Satz, der angesprochen werden soll, wird durch r spezifiziert. Erfolgt die Eingabe/Ausgabe unter Steuerung einer FORMAT-Vereinbarung, so ist unter b die Statementnummer derselben anzugeben.

g) READ b, liste *)
PUNCH b. liste *)
PRINT b, liste *)
Diese Anweisungen sind äquivalent den folgenden Anweisungen:
READ (5, b) liste
WRITE (7, b) liste
WRITE (6, b) liste

h) END FILE a
REWIND a
BACKSPACE a

Durch a wird wiederum die Referenznummer der Datei spezifiziert; die Datei sollte auf einem Magnetband liegen.

Anmerkungen: 1) Die mit einem Sternzeichen versehenen Statements haben nur für die volle Sprache Gültigkeit.
2) Unter liste ist stets eine Datenliste zu verstehen (siehe unter 2.)

4) Statements für Unterprogramme

a) $a(p_1, \cdots, p_n) = b$
Hierbei bedeutet a den Namen einer Funktion; die p_i bezeichnen die Parameter, die nichtindizierte Variable darstellen müssen. Durch b wird ein beliebiger Ausdruck angezeigt.

b) FUNCTION $a(p_1, \cdots, p_n)$
typ FUNCTION $a(p_1, \cdots, p_n)$
Durch typ wird hierbei der Typ des Funktionswertes angezeigt; das kann INTEGER, REAL, COMPLEX oder LOGICAL (die letzten beiden jedoch nur bei der vollen Sprache) sein. Wird der Typ des Funktionswertes nicht spezifiziert, so ist der Typ durch den ersten Buchstaben des Funktionsnamens a festgelegt. Die p_i bedeuten die Parameter der Funktion.
Diese Vereinbarung muß als erstes Statement eines Funktionsunterprogrammes erscheinen.

c) SUBROUTINE $a(p_1, \cdots, p_n)$
Der Name des Unterprogrammes wird durch a festgelegt, die p_i geben in bekannter Weise die Parameter an. Als Parameter können Variable oder Bereiche gewählt werden. In der vollen Sprache sind auch Sternzeichen als Parameter zulässig.

d) BLOCK DATA *)
Dieses Statement muß als erstes Statement eines Blockdatenunterprogrammes erscheinen.

e) DATA $v_1, \cdots, v_n/r_1{*}d_1, \cdots, r_m{*}d_m/\cdots$ *)
Hierbei bedeuten die v_i Variable, die zusätzlich konstante Indizes aufweisen können, d.h. Bereichselemente sein können. Durch die v_i können auch Bereiche spezifiziert werden. Die r_i stellen Wiederholungsfaktoren dar; werden sie weggelassen, so wird an ihrer Stelle eine 1 angenommen. Zwischen den Schrägstrichen müssen genau so viele Werte aufgeführt werden, wie Variable bzw. Bereichselemente vor den Schrägstrichen stehen.
Durch DATA-Statements werden Variablen bzw. Bereichselementen Anfangswerte zugewiesen. Die Anfangswerte sind durch die d_i angedeutet.

f) CALL $a(v_1, \cdots, v_n)$
Hierbei wird durch a der Name des aufzurufenden Subroutine-Unterprogrammes festgehalten; durch die v_i werden die Argumente spezifiziert, diese dürfen Konstanten, Variablen, Bereiche oder Unterprogrammnamen sein. Die Argumente müssen jedoch in der Anzahl, der Reihenfolge und dem Typ mit den Parametern in der SUBROUTINE-Vereinbarung harmonieren.
Es erfolgt eine Verzweigung zum aufgerufenen Unterprogramm; diesem werden außerdem die angegebenen Argumente übergeben, die an die Stelle der Parameter treten.

g) RETURN
RETURN i *)
Hierbei stellt i eine ganzzahlige Konstante oder Variable dar. Der Wert von i bezieht sich auf die i-te Statementnummer in der Parameterliste des SUBROUTINE-Statements.
Wenn i nicht angegeben ist, dann wird die Programmablaufsteuerung dem nächsten Statement (nach der CALL-Anweisung) der aufrufenden Programmeinheit übergeben. Die RETURN-Anweisung muß die letzte Anweisung jedes Funktions- bzw. jedes Subroutine-Unterprogrammes sein.

h) ENTRY $a(p_1, \cdots, p_n)$
Diese Vereinbarung erfüllt die gleiche Funktion wie die Vereinbarungen FUNCTION bzw. SUBROUTINE. Durch ENTRY können weitere Eingangsstellen in Unterprogramme festgelegt werden.

i) EXTERNAL $a_1, \cdots$
Hierbei bedeuten die a_i die Namen von Funktions- oder von Subroutine-Unterprogrammen, die als Argumente beim Aufruf von anderen Unterprogrammen erscheinen sollen, in den aufgerufenen Programmeinheiten jedoch nicht selbst explizit aufgerufen werden.

Anmerkung: Die mit einem Sternzeichen versehenen Statements haben nur für die volle Sprache Gültigkeit.

5) Spezifikationsvereinbarungen

a) DIMENSION $v_1, v_2, \cdots$
Jedes v_i stellt den Namen eines Bereiches dar; dieser muß durch positive ganzzahlige Indizes gekennzeichnet sein. Diese ganze Zahlen geben die Maximalwerte an, die der betreffende Index im Programm annehmen kann.

b) EQUIVALENCE $(a, b, \cdots), (c, d, \cdots), \cdots$
Durch $a, b, \cdots, c, d, \cdots$ werden Variable dargestellt. Diese können normal indiziert sein, sie dürfen aber auch durch einen einzigen Index gekennzeichnet sein. Dieser spiegelt dann die relative Stellung des betreffenden Bereichselementes im Bereich wider. Jedes Klammernpaar umschließt diejenigen Variablen bzw. Bereichselemente, denen der gleiche Hauptspeicherplatz zugeordnet werden soll.

c) COMMON $v_1, v_2, \cdots, v_n/r/u_1, u_2, \cdots, u_m/s/\cdots$
Jedes v_i bzw. u_i stellt den Namen einer Variablen dar; diese können durch Dimensionsinformationen ergänzt sein, die die gleiche Gestalt aufweisen wie bei der DIMENSION-Vereinbarung. Durch r bzw. s werden die Namen von Speicherblöcken innerhalb des COMMON-Speicherbereiches festgehalten. Variablen innerhalb von Listen, denen kein von Schrägstrichen umgebener Speicherblockname voransteht, gehören zum unbenannten Speicherblock des COMMON-Bereiches. Die Angabe von Blocknamen ist nur in der vollen Sprache möglich.

d) FORMAT $(s_1, s_2, \cdots, s_n)$
Hierbei stellt jedes s_i eine Formatangabe dar. Durch diese ist die erforderliche Umwandlung und das Format der Werte bestimmt, die den in der Datenliste der zugehörigen E/A-Anweisung spezifizierten Variablen zugewiesen sind.

e) NAMELIST $/x/v_1, v_2, \cdots, v_n/y/u_1, u_2, \cdots, u_m/z/\cdots$ *)
Jeder von Schrägstrichen umgebener Name, hier durch x, y, z, $\cdots$ dargestellt, ist der Name einer sogenannten Namensliste (NAMELIST). Alle nachfolgenden Variablen sind Mitglieder der betreffenden Liste. In einem Statement dürfen beliebig viele Namenslisten festgelegt werden.

f) IMPLICIT typ*s$(a_1, a_2, \cdots, a_n)$, typ*s$(\cdots)$, $\cdots$ *)
Durch typ wird entweder INTEGER, REAL, COMPLEX oder LOGICAL angezeigt; die Angabe von s ist freigestellt, durch s kann eine Länge von 1, 2, 4, 8 oder 16 Bytes spezifiziert werden. Wird die Angabe von s weggelassen, so wird INTEGER*4, REAL*4, COMPLEX*8 bzw. LOGICAL*4 angenommen. Durch die a_i werden entweder Buchstaben oder eine Buchstabengruppe festgehalten. Buchstaben bzw. Buchstabengruppen müssen durch Kommas voneinander getrennt werden. Eine Buchstabengruppe ist wie folgt niederzuschreiben: Der erste und der letzte Buchstabe der Gruppe sind durch ein Minuszeichen (Bindestrich) miteinander zu verbinden. Alle Variablen, deren Namen mit dem angegebenen Buchstaben beginnt bzw. deren Namen mit einem Buchstaben aus der angegebenen Buchstabengruppe beginnt, werden implizit als zu dem Typ zugehörig betrachtet, der durch das vorangehende typ*s spezifiziert ist. Eine solche implizite Typerklärung kann durch eine explizite Typerklärung zunichte gemacht werden.

g) typ*s a_1*$s_1(k_1)/x_1/, \cdots$
Bei diesen expliziten Typvereinbarungen wird durch typ der Typ der betreffenden Variablen festgehalten, das kann INTEGER, REAL, COMPLEX oder LOGICAL sein (die Angabe von COMPLEX bzw. LOGICAL ist nur in der vollen Sprache möglich). Die Angabe von s ist wie bei der IMPLICIT-Vereinbarung freigestellt: s besitzt hier die gleiche Bedeutung wie bei dieser Vereinbarung. Durch die a_i werden die Namen derjenigen Variablen festgehalten, deren Typ festgelegt werden soll. Die k_i stellen Dimensionsinformationen dar; diese müssen in derselben Gestalt wie bei der DIMENSION-Vereinbarung auftreten. Der Wert von s gilt für alle Variablen, für die ein s_i nicht gesondert niedergeschrieben ist. Wird die Angabe von s bzw. von s_i weggelassen, so wird implizit die gleiche Länge für die Variablen angenommen wie bei IMPLICIT-Statements.

Wird $\underline{x_i}$ niedergeschrieben, so wird der betreffenden Variablen bzw. den betreffenden Bereichselementen, der Anfangswert $\underline{x_i}$ zugewiesen. Die $\underline{x_i}$ können bei Bereichen selbstverständlich auch mit einem Wiederholungsfaktor versehen werden bzw. auch Listen von Konstanten darstellen. Die Angabe von Anfangswerten ist nur in der vollen Sprache möglich.

h) DOUBLE PRECISION $a_1(k_1), \cdots$
Hierbei bedeuten die $\underline{k_i}$ Dimensionsinformationen, die das gleiche Aussehen haben müssen wie bei den DIMENSION-Vereinbarungen. Jedes $\underline{a_i}$ stellt den Namen einer Variablen dar, deren Typ zu REAL*8 festgelegt werden soll.

Anmerkungen: Die mit einem Sternzeichen versehenen Statements haben nur für die volle FORTRAN-Sprache Gültigkeit.

7. Funktionen der FORTRAN-Sprache (offene Unterprogramme, auch „In-Line-Funktionen" genannt)

Beschreibung der Funktion	Funktionswert	Funktionsnamen	Anzahl der Argumente	Typ 3) der Argumente	Typ 3) des Funktionswertes
Bestimmung des absoluten Betrages	$\|x\|$	IABS	1	I4	I4
		ABS	1	R4	R4
		DABS	1	R8	R8
Absolutbetrag (Modul) von komplexen Zahlen	$\sqrt{a^2 + b^2}$ von $x = a + bi$	CABS	1 1)	C8	R8
		CDABS		C16	C8
Umwandlung in die Gleitpunktdarstellung	x in Gleitpunktdarstellung	FLOAT	1	I4	R4
		DFLOAT	1	I4	R8
Umwandlung in die Festpunktdarstellung	x in Festpunktdarstellung	IFIX	1	R4	I4
		HFIX	1 1)	R4	I2
Vorzeichenübertragung	$sgn(x_2) * \|x_1\|$ 2)	SIGN	2	R4	R4
		ISIGN	2	I4	I4
		DSIGN	2	R8	R8
Positive Differenz	$x_1 - \min(x_1, x_2)$	DIM	2	R4	R4
		IDIM	2	I4	I4
Umwandlung in einfache Genauigkeit		SNGL	1	R8	R4
Umwandlung in doppelte Genauigkeit		DBLE	1	R4	R8
Bildung des Realteils komplexer Zahlen		REAL	1 1)	C8	R4
Bildung des Imaginärteils komplexer Zahlen		AIMAG	1 1)	C8	R4
Bildung komplexer Zahlen	$x_1 + x_2 i$	CMPLX	2 1)	R4	C8
		DCMPLX	2 1)	R8	C16
Bildung von konjugiert komplexen Zahlen	$a - bi$ von $x = a + bi$	CONJG	1 1)	C8	C8
		DCONJG	1 1)	C16	C16

Anmerkungen:
1) Diese Funktionen sind nur in der vollen Sprache definiert
2) Für die Funktion sgn gilt folgende Definition

$$sgn(x) = \begin{cases} +1 & \text{für } x > 0 \\ 0 & \text{für } x = 0 \\ -1 & \text{für } x < 0 \end{cases}$$

3) Es bedeuten:

I2	INTEGER*2
I4	INTEGER*4
R4	REAL*4
R8	REAL*8
C8	COMPLEX*8
C16	COMPLEX*16

8. Funktionen der FORTRAN-Sprache (geschlossene Unterprogramme, auch „Out-of-Line-Funktionen" genannt)

Beschreibung der Funktion	Funktionswert	Funktionsnamen	Anzahl der Argumente		Typ 2) der Argumente	Typ 2) des Funktionswertes
Exponentialfunktion	e^x bzw. exp(x)	EXP	1		R4	R4
		DEXP	1		R8	R8
		CEXP	1	1)	C8	C8
		CDEXP	1	1)	C16	C16
Logarithmusfunktion (Basis e)	ln(x)	ALOG	1		R4	R4
		DLOG	1		R8	R8
		CLOG	1	1)	C8	C8
		CDLOG	1	1)	C16	C16
Logarithmusfunktion (Basis 10)	lg(x)	ALOG10	1		R4	R4
		DLOG10	1		R8	R8
		CLOG10	1	1)	C8	C8
		CDLG10	1	1)	C16	C16
Arcustangensfunktion	arc tan (x)	ATAN	1		R4	R4
		DATAN	1		R8	R8
Arcustangensfunktion	arc tan (x_1/x_2)	DATAN2	2	1)	R8	R8
Sinusfunktion	sin (x)	SIN	1		R4	R4
		DSIN	1		R8	R8
		CSIN	1	1)	C8	C8
		CDSIN	1	1)	C16	C16
Cosinusfunktion	cos (x)	COS	1		R4	R4
		DCOS	1		R8	R8
		CCOS	1	1)	C8	C8
		CDCOS	1	1)	C16	C16
Quadratwurzelfunktion	$\sqrt{x}$	SQRT	1		R4	R4
		DSQRT	1		R8	R8
		CSQRT	1	1)	C8	C8
		CDSQRT	1	1)	C16	C16
Hyperbeltangensfunktion	tanh (x)	TANH	1		R4	R4
		DTANH	1		R8	R8
Restbildung	$x_1 - [x_1/x_2] * x_2$ 4)	MOD	2		I4	I4
		AMOD	2		R4	R4
		DMOD	2		R8	R8
Bestimmung des Maximalwertes	max $(x_1, x_2, \ldots)$	AMAX0	≥ 2		I4	R4
		AMAX1	≥ 2		R4	R4
		MAX0	≥ 2		I4	I4
		MAX1	≥ 2		R4	I4
		DMAX1	≥ 2		R8	R8
Bestimmung des Minimalwertes	min $(x_1, x_2, \ldots)$	AMIN0	≥ 2		I4	R4
		AMIN1	≥ 2		R4	R4
		MIN0	≥ 2		I4	I4
		MIN1	≥ 2		R4	I4
		DMIN1	≥ 2		R8	R8
Abbrechen der Ziffernfolge einer Zahl (Abschneiden)	sgn (x) * [| x |] 3) 4)	INT	1		R4	I4
		AINT	1		R4	R4
		IDINT	1		R8	I4

Anmerkungen:

1) Diese Funktionen sind nur in der vollen Sprache definiert.

2) Es bedeuten

I4	INTEGER*4
R4	REAL*4
R8	REAL*8
C8	COMPLEX*8
C16	COMPLEX*16

3) Es ist

$$\text{sgn}(x) = \begin{cases} +1 & \text{für } x<0 \\ 0 & \text{für } x=0 \\ -1 & \text{für } x>0 \end{cases}$$

4) Es bedeutet:
[x] → größter ganzzahliger Wert von x

Anhang G: Reservierte Wörter in COBOL

Die reservierten Wörter von COBOL sind in alphabetischer Reihenfolge aufgeführt. Es ist nicht vermerkt, in welchem Teil eines Programmes sie auftreten dürfen. Nicht alle Wörter sind in den Kapiteln 17 bis 19 besprochen worden.

A

ACCEPT
ACCESS
ACTUAL
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALPHANUMERIC
ALPHANUMERIC-
EDITED
ALTER
ALTERNATE
AND
APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR

B

BASIS
BEFORE
BEGINNING
BLANK
BLOCK
BOTTOM
BY

C

CALL
CANCEL
CBL
CD
CF
CH
CHANGED
CHARACTER
CHARACTERS
CLOCK-UNITS
CLOSE
COBOL
CODE
COLUMN
COM-REG
COMMA
COMMUNICATION
COMP
COMP-1
COMP-2
COMP-3
COMP-4
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTATIONAL-4
COMPUTE
CONFIGURATION
CONSOLE
CONSTANT
CONTAINS
CONTROL
CONTROLS
COPY
CORE-INDEX
CORR
CORRESPONDING
COUNT
CSP
CURRENCY
CURRENT-DATE
CYL-INDEX
CYL-OVERFLOW
CØ1
CØ2
CØ3
CØ4
CØ5
CØ6
CØ7
CØ8
CØ9
C1Ø
C11
C12

D

DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DAY-OF-WEEK
DE
DEBUG
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUGGING
DECIMAL-POINT
DELARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DEPTH
DESCENDING
DESTINATION
DETAIL
DISABLE
DISP
DISPLAY
DISPLAY-ST
DISPLAY-n
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC

E

EGI
EJECT
ELSE
EMI
ENABLE
END
END-OF-PAGE
ENDING
ENTER
ENTRY
ENVIRONMENT
EOP
EQUAL
EQUALS
ERROR
ESI
EVERY
EXAMINE
EXCEEDS
EXCEPTION
EXHIBIT
EXIT
EXTEND
EXTENDEN-SEARCH

F

FD
FILE
FILE-CONTROL
FILE-LIMIT
FILE-LIMITS
FILLER
FINAL
FIRST
FOOTING
FOR
FROM

G

GENERATE
GIVING
GO
GOBACK
GREATER
GROUP

H

HEADING
HIGH-VALUE
HIGH-VALUES
HOLD

I

I-O
I-O-CONTOL
ID
IDENTIFICATION
IF
IN
INDEX
INDEX-n
INDEXED
INDICATE
INITIAL
INITIALIZE
INITIALE
INPUT
INPUT-OUTPUT
INSERT
INSPECT

INSTALLATION
INTO
INVALID
IS

J

JUST
JUSTIFIED

K

KEY
KEYS

L

LABEL
LABEL-RETURN
LAST
LEADING
LEAVE
LEFT
LENGTH
LESS
LIBRARY
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKAGEÄ
LOCK
LOW-VALUE
LOW-VALUES
LOWER-BOUND
LOWER-BOUNDS

M

MASTER-INDEX
MEMORY
MERGE
MESSAGE
MODE
MODULES
MORE-LABELS
MOVE
MULTIPLE
MULTIPLY

N

NAMED
NEGATIVE
NEXT
NO
NOMINAL
NOT
NOTE
NSTD-REELS
NUMBER
NUMERIC
NUMERIC-EDITED

O

OBJECT-COMPUTER
OBJECT-PROGRAM
OCCURS
OF
OFF
OH
OMITTED
ON
OPEN
OPTIONAL
OR
ORGANIZATION
OTHERWISE
OUTPUT
OV
OVERFLOW

P

PAGE
PAGE-COUNTER
PASSWORD
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIONING
POSITIVE
PREPARED
PRINT-SWITCH
PRINTING
PRIORITY
PROCEDURE
PROCEDURES
PROCEED
PROCESS
PROCESSING
PROGRAM
PROGRAM-ID

Q

QUEUE
QUOTE
QUOTES

R

RANDOM
RANGE
RD
READ
READY
RECEIVE
RECORD
RECORD-OVERFLOW
RECORDING
RECORDS
REDEFINES
REEL
REFERENCES
RELATIVE
RELEASE
RELOAD
REMAINDER
REMARKS
REMOVAL
RENAMES
REORG-CRITERIA
REPLACING
REPORT
REPORTING
REPORTS
REREAD
RERUN
RESERVE
RESET
RETURN
RETURN-CODE
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN

S

SA
SAME
SD
SEARCH
SECTION
SECURITY
SEEK
SEGMENT
SEGMENT-LIMIT
SELECT
SELECTED
SEND
SENTENCE
SEPARATE
SEQUENTIAL
SERVICE
SET
SIGN
SIZE
SKIP1
SKIP2
SKIP3
SORT
SORT-CORE-SIZE
SORT-FILE-SIZE
SORT-MERGE
SORT-MESSAGE
SORT-MODE-SIZE
SORT-RETURN
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
START
STATUS
STOP
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUBTRACT
SUM
SUPERVISOR
SUPPRESS
SUSPEND
SYMBOLIC
SYNC
SYNCHRONIZED
SYSIN
SYSIPT
SYSLST
SYSOUT
SYSPCH
SYSPUNCH
SØ1
SØ2

T

TABLE
TALLY
TALLYING
TAPE
TERMINAL
TERMINATE
TEXT
THAN
THEN
THROUGH
THRU
TIME
TIME-OF-DAY
TIMES
TO
TOP
TOTALED
TOTALING
TRACE
TRACK

TRACK-AREA
TRACK-LIMIT
TRACKS
TRAILING
TRANSFORM
TYPE

U

UNEQUAL
UNIT
UNSTRING
UNTIL
UP
UPON
UPPER-BOUND
UPPER-BOUNDS
UPSI-Ø
UPSI-1
UPSI-2
UPSI-3
UPSI-4
UPSI-5
UPSI-6
UPSI-7
USAGE
USE
USING

V

VALUE
VALUES
VARYING

W

WHEN
WITH
WORDS
WORKING-STORAGE
WRITE
WRITE-ONLY
WRITE-VERIFY

Z

ZERO
ZEROES
ZEROS

Anhang H: Kenngrößen der SEDZ

1. Spurkapazitäten

Maximale Anzahl von Bytes pro Satz ohne Schlüssel							Anzahl der Datensätze pro Spur	Maximale Anzahl von Bytes pro Satz mit Schlüssel						
23Ø3	23Ø5-1	23Ø5-2	2311	2314	333Ø	334Ø		23Ø3	23Ø5-1	23Ø5-2	2311	2314	333Ø	334Ø
4 892	14 136	14 66Ø	3 625	7 294	13 Ø3Ø	8 638	1	4 854	13 934	14 569	3 6Ø2	7 249	12 974	8 293
2 392	6 853	7 231	1 74Ø	3 521	6 447	4 1ØØ	2	2 354	6 651	7 14Ø	1 72Ø	3 476	6 391	4 Ø25
1 558	4 425	4 754	1 131	2 298	4 253	2 678	3	1 52Ø	4 223	4 663	1 111	2 254	4 197	2 6Ø3
1 142	3 211	3 516	83Ø	1 693	3 156	1 966	4	1 1Ø4	3 211	3 425	811	1 649	3 1ØØ	1 891
892	2 483	2 773	651	1 332	2 498	1 54Ø	5	854	2 281	2 682	632	1 288	2 442	1 465
725	1 997	2278	532	1 Ø92	2 Ø59	1 255	6	687	1 795	2 187	512	1 Ø49	2 ØØ3	1 18Ø
6Ø6	1 65Ø	1 924	447	921	1 745	1 Ø52	7	568	1 448	1 833	428	878	1 689	977
517	1 39Ø	1 659	384	793	1 51Ø	899	8	479	1 188	1 568	364	75Ø	1 454	824
447	1 188	1 452	334	694	1 327	781	9	4Ø9	986	1 361	315	65Ø	1 271	7Ø6
392	1 Ø26	1 287	295	615	1 181	686	1Ø	354	824	1 196	275	571	1 125	611
346	894	1 152	263	55Ø	1 Ø61	6Ø8	11	3Ø8	692	1 Ø61	244	5Ø6	1 ØØ5	533
3Ø8	783	1 Ø4Ø	236	496	962	544	12	27Ø	581	949	217	452	9Ø6	469
276	69Ø	944	213	45Ø	877	489	13	238	488	853	194	4Ø7	821	414
249	61Ø	863	193	411	8Ø5	442	14	211	4Ø8	772	174	368	749	367
225	541	792	177	377	742	4Ø2	15	187	339	7Ø1	158	333	686	327
2Ø4	48Ø	73Ø	162	347	687	366	16	166	278	639	143	3Ø4	631	291
186	426	676	149	321	639	335	17	148	224	585	13Ø	277	583	26Ø
169	379	627	138	298	596	3Ø7	18	131	177	536	119	254	54Ø	232
155	336	584	127	276	557	282	19	117	134	493	1Ø8	233	5Ø1	2Ø7
142	298	544	118	258	523	259	2Ø	1Ø4	96	453	99	215	467	184

Maximale Anzahl von Bytes pro Satz ohne Schlüssel							Anzahl der Datensätze pro Spur	Maximale Anzahl von Bytes pro Satz mit Schlüssel						
23Ø3	23Ø5-1	23Ø5-2	2311	2314	333Ø	334Ø		23Ø3	23Ø5-1	23Ø5-2	2311	2314	333Ø	334Ø
13Ø	263	5Ø9	1Ø9	241	491	239	21	92	61	418	9Ø	198	435	164
119	232	477	1Ø2	226	463	22Ø	22	81	3Ø	386	82	183	4Ø7	145
1Ø9	2Ø3	448	95	211	437	2Ø4	23	71	1	357	76	168	381	129
1ØØ	176	421	88	199	413	188	24	62		33Ø	69	156	357	113
92	152	396	82	187	391	174	25	54		3Ø5	63	144	335	99
84	13Ø	373	77	176	371	161	26	46		282	58	133	315	86
77	1Ø9	352	72	166	352	149	27	39		261	53	123	296	74
7Ø	9Ø	332	67	157	335	137	28	32		241	48	114	279	62
64	72	314	63	148	318	127	29	26		223	44	1Ø5	262	52
58	55	297	59	139	3Ø3	117	3Ø	2Ø		2Ø6	4Ø	96	247	42
53	39	281	55	132	289	1Ø8	31	15		19Ø	36	89	233	33
48	25	266	52	125	276	99	32	1Ø		175	33	82	22Ø	24
43	11	252	48	118	263	91	33	5		161	29	75	2Ø7	16
39		239	45	112	252	84	34	1		148	26	69	196	9
34		226	42	1Ø6	241	76	35			135	23	63	185	1
3Ø		214	4Ø	1ØØ	23Ø	7Ø	36			123	2Ø	57	174	
27		2Ø3	37	94	22Ø	63	37			112	18	51	164	
23		193	35	9Ø	211	57	38			1Ø2	16	46	155	
2Ø		182	32	85	2Ø2	51	39			91	13	42	146	
17		173	3Ø	8Ø	194	46	4Ø			82	11	37	138	
13		164	27	76	186	41	41			73	8	33	13Ø	
11		155	25	72	178	36	42			64	6	29	122	
8		147	23	69	171	31	43			56	4	25	115	
5		139	21	65	164	26	44			48	2	22	1Ø8	
3		132	2Ø	61	157	22	45			41		18	1Ø1	

Maximale Anzahl von Bytes pro Satz ohne Schlüssel							Anzahl der Datensätze pro Spur	Maximale Anzahl von Bytes pro Satz mit Schlüssel						
23Ø3	23Ø5-1	23Ø5-2	2311	2314	333Ø	334Ø		23Ø3	23Ø5-1	23Ø5-2	2311	2314	333Ø	334Ø
		125	19	57	151	18	46			34		14	95	
		118	17	54	145	14	47			27		11	89	
		111	15	51	139	1Ø	48			2Ø		8	83	
		1Ø5	14	47	133	7	49			14		4	77	
		99	12	45	128	3	5Ø			8		1	72	
		93	11	43	123		51			2			67	
		87	9	4Ø	118		52						62	
		82	8	37	113		53						57	
		77	7	34	1Ø8		54						52	
		72	6	32	1Ø4		55						48	
		67	4	3Ø	1ØØ		56						44	
		62	3	27	95		57						39	
		58	2	25	91		58						35	
		53	1	23	88		59						32	
		49		22	84		6Ø						28	
		45		2Ø	8Ø		61						24	
		41		18	77		62						21	
		37		16	73		63						17	
		34		14	7Ø		64						14	
		3Ø		12	67		65						11	
		27		11	64		66						8	
		23		9	61		67						5	
		2Ø		7	58		68						2	
		17		6	55		69							
		14		4	53		7Ø							

Maximale Anzahl von Bytes pro Satz ohne Schlüssel						Anzahl der Datensätze pro Spur	Maximale Anzahl von Bytes pro Satz mit Schlüssel						
2303	2305-1	2305-2	2311	2314	3330		2303	2305-1	2305-2	2311	2314	3330	3340
		11		3	50	71							
		8		1	47	72							
		5			45	73							
		2			42	74							
					40	75							
					38	76							
					35	77							
					33	78							
					31	79							
					29	80							
					27	81							
					25	82							
					23	83							
					21	84							
					19	85							
					18	86							
					16	87							
					14	88							
					12	89							
					11	90							
					9	91							
					8	92							
					6	93							
					5	94							
					3	95							
					2	96							

2. Übersicht über die Kenngrößen der SEDZ

Gerät		Zylinder		Schreib/Lese-Köpfe	Bytes/Spur	Kenngrößen			Zugriffszeit in ms	
		Primärzyl.	Ersatzzyl.			a	b	c	Durchschnittliche Z.	Maximale Z.
T	2303	80	0	10	4 892	108	38	1	8,6	17,50
P	2305-1	48	1	8	14 136	430	202	1	2,5	5,10
P	2305-2	96	1	8	14 660	198	91	1	5,0	10,25
P	2311	200	3	10	3 625	61	20	537/512	75,0	135,00
P	2314	200	3	20	7 294	101	45	534/512	60,0 *	130,00 *
P	3330-1	404	7	19	13 030	135	56	1	30,0	55,00
P	3330-11	808	7	19	13 030	135	56	1	30,0	55,00
P	3340-35	348	1	12	8 368	167	75	1	25,0	50,00
P	3340-70	696	2	12	8 368	167	75	1	25,0	50,00

**Die angegebenen Werte beziehen sich auf die Baureihen A und B; beim Modell 1 der IBM 2314 sind die Zugriffszeiten diegleichen wie bei der IBM 2311*

Anmerkungen: 1) Für die Bestimmung der für einen Satz erforderlichen Bytes z auf einer Spur gilt die nachstehende Formel

$$z = \left\{\begin{matrix} a \\ 0 \end{matrix}\right\} + \left\{\begin{matrix} b \\ 0 \end{matrix}\right\} + \left\{\begin{matrix} c \\ 1 \end{matrix}\right\} \cdot \left(\left\{\begin{matrix} s \\ 0 \end{matrix}\right\} + \left\{\begin{matrix} d \\ 1 \end{matrix}\right\} \right)$$

Hierbei sind a, b, c der obigen Tabelle zu entnehmen, s bedeutet die Schlüsselfeldlänge und d die Datenfeldlänge

2) Für den letzten Satz einer Spur ist für a der Wert 0 und für c der Wert 1 einzusetzen; bei einem Satz ohne Schlüssel ist für b und s der Wert 0 einzusetzen

3) Bei einem Dateiendesatz (EOF-Record) ist für d der Wert 1 einzusetzen

Anhang I: Steuerzeichen

Diese Übersicht enthält die ASA-Steuerzeichen (auch USASI- oder ANS-Steuerzeichen genannt).

a) für Drucker

Steuer-zeichen	Funktion
b [1])	Einzeiliger Zeilentransport vor dem Drucken
0	Zweizeiliger Zeilentransport vor dem Drucken
-	Dreizeiliger Zeilentransport vor dem Drucken
+	Kein Zeilentransport vor dem Drucken
1	Vorschub auf Kanal 1 vor dem Drucken
2	Vorschub auf Kanal 2 vor dem Drucken
3	Vorschub auf Kanal 3 vor dem Drucken
4	Vorschub auf Kanal 4 vor dem Drucken
5	Vorschub auf Kanal 5 vor dem Drucken
6	Vorschub auf Kanal 6 vor dem Drucken
7	Vorschub auf Kanal 7 vor dem Drucken
8	Vorschub auf Kanal 8 vor dem Drucken
9	Vorschub auf Kanal 9 vor dem Drucken
A	Vorschub auf Kanal 10 vor dem Drucken
B	Vorschub auf Kanal 11 vor dem Drucken
C	Vorschub auf Kanal 12 vor dem Drucken

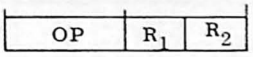
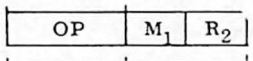
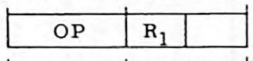
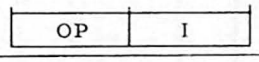
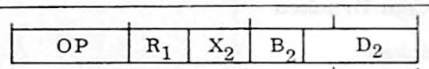
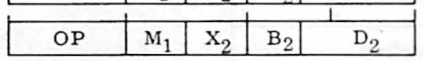
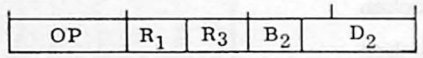
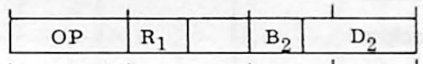
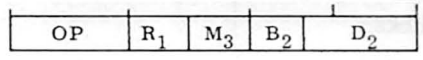
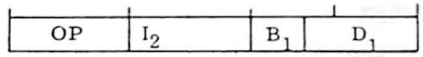
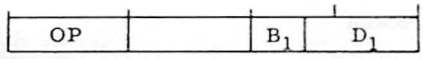
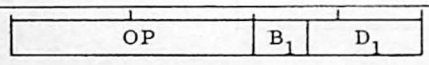
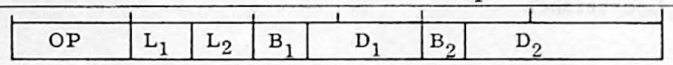
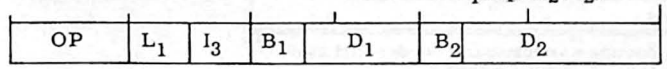
Anmerkung: b bedeutet hier das Leerzeichen

b) für Kartenstanzer

Steuer-zeichen	Funktion	Bemerkungen für das Betriebssystem DOS
V	Ablage im Ablagefach 1	a) Zweites Kartenmagazin bei der IBM 2560: Ablagefach 5 bei DTFDI-Dateien b) Zweites Kartenmagazin bei der IBM 5426: Ablagefach 4 bei DTFDI-Dateien
W	Ablage im Ablagefach 2	Zweites Kartenmagazin bei IBM 2560 und IBM 5425: Ablagefach 3 bei DTFDI-Dateien
X	Ablage im Ablagefach 3	Bei IBM 2560 und IBM 5425 (nur bei DTFCD-Dateien)
Y	Ablage im Ablagefach 4	Bei IBM 2560 und IBM 5425 (nur bei DTFCD-Dateien)
Z	Ablage im Ablagefach 5	Bei IBM 2560 (nur bei DTFCD-Dateien)

Anhang J: Übersicht über die Assembliersprache (AL)

1. Instruktionsformate

Maschinenformat	Assembliersprache	Instruktionen
RR 	R_1, R_2	Alle Instruktionen im RR-Format mit Ausnahme von SPM, SVC und BCR
RR 	M_1, R_2	BCR
RR 	R_1	SPM
RR 	I	SVC
RX 	$R_1, D_2(X_2, B_2)$ $R_1, S_2(X_2)$ R_1, S_2	Alle Instruktionen im RX-Format mit Ausnahme von BC
RX 	$M_1, D_2(X_2, B_2)$ $M_1, S_2(X_2)$ M_1, S_2	BC
RS 	$R_1, R_3, D_2(B_2)$ R_1, R_3, S_2	BXH, BXLE, STM, LM, LCTL, STCTL
RS 	$R_1, D_2(B_2)$ R_1, S_2	Alle Verschiebungsinstruktionen
RS 	$R_1, R_3, D_2(B_2)$ R_1, M_3, S_2	ICM, STCM, CLM
SI 	$D_1(B_1), I_2$ S_1, I_2	NI, CLI, OI, XI, TM, MVI, RDD, WRD
SI 	$D_1(B_1)$ S_1	SSM, LPSW, TS, TCH
S 	$D_1(B_1)$ S_1	Alle Instruktionen im S-Format
SS 	$D_1(L_1, B_1), D_2(L_2, B_2)$ $D_1(L_1, B_1), S_2(L_2)$ $S_1(L_1), D_2(L_2, B_2)$ $S_1(L_1), S_2(L_2)$	AP, SP, MP, DP CP, ZAP, MVO, PACK, UNPK
SS 	$D_1(L_1, B_1), D_2(B_2), I_3$ $D_1(L_1, B_1), S_2, I_3$ $S_1(L_1), D_2(B_2), I_3$ $S_1(L_1), S_2, I_3$	SRP

Es bedeuten:

a) OP	Operationsschlüssel
b) R_1, R_2, R_3	Absolute Ausdrücke; diese bezeichnen Register
c) D_1, D_2	Absolute Ausdrücke kleiner als 4096; diese geben relative Adressen an
d) B_1, B_2	Absolute Ausdrücke; diese spezifizieren Basisregister
e) X_2	Absolute Ausdrücke; diese spezifizieren Indexregister
f) L_1, L_2	Absolute Ausdrücke; diese spezifizieren Feldlängen zwischen 1 und 16
g) L	Absolute Ausdrücke; diese spezifizieren Feldlängen zwischen 1 und 256
h) I, I_2	Absolute Ausdrücke; diese geben Direktdaten zwischen 0 und 255 einschließlich an
i) S_1, S_2	Absolute oder verschiebliche Ausdrücke; diese spezifizieren Hauptspeicheradressen
j) M_1, M_3	Absolute Ausdrücke; diese spezifizieren Masken mit vier Bitstellen
k) I_3	Absoluter Ausdruck; dieser spezifiziert die Rundungsziffer

2. Konstanten

Typ	Implizite Länge (in Bytes)	Begrenzungs-zuordnung	Länge in Bytes	Festlegung durch	Mehrfache Konstanten	Abschneiden Auffüllen
C	1)	Byte	DC:0, 1···256 DS:0, 1···65536	Zeichen	Nein	nach rechts
B	1)	Byte	0, 1...256	Dualziffern	Nein	nach links
X	1)	Byte	DC:0, 1···256 DS:0, 1···65536	Sedezimalziffern	Nein	nach links
H	2	Halbwort	0, 1···8	Dezimalziffern	Ja	nach links
F	4	Wort	0, 1···8	Dezimalziffern	Ja	nach links
E	4	Wort	0, 1···8	Dezimalziffern	Ja	nach rechts
D	8	Doppelwort	0, 1···8	Dezimalziffern	Ja	nach rechts
Z	1)	Byte	0, 1···16	Dezimalziffern	Ja	nach links
P	1)	Byte	0 1···16	Dezimalziffern	Ja	nach links
A	4	Wort	2)	Beliebiger Ausdruck	Ja	nach links
Y	2	Halbwort	3)	Beliebiger Ausdruck	Ja	nach links
V	4	Wort	4)	Verschiebliches Symbol	Ja	nach links
S	2	Halbwort	2	Absoluter Ausdruck; Verschieblicher Ausdruck zwei absolute Ausdrücke in der Form a (a)	Ja	keine
L	16	Doppelwort	0, 1···16	Dezimalziffern	Ja	nach rechts

Bemerkungen:

1) So viele Bytes wie benötigt werden
2) Verschiebliche Konstanten vom A-Typ müssen eine Länge von 3 oder 4 Bytes aufweisen; absolute Konstanten des gleichen Typs können Längen zwischen 0, 1 und 4 Bytes aufweisen
3) Verschiebliche Konstanten von Y-Typ müssen eine Länge von 2 Bytes aufweisen; absolute Konstanten des gleichen Typs können Längen zwischen 0, 1 und 2 Bytes aufweisen
4) Konstanten vom V-Typ müssen 3 oder 4 Bytes lang sein
5) Wertebereich der Exponenten bei Konstanten des H-, F-, E- und D-Types: -85 bis +75
6) Wertebereich des Skalenfaktors für Konstanten des H- und F-Typs: -187 bis +346
   Wertebereich des Skalenfaktors für Konstanten des D- und E-Typs: 0 bis 14
   Wertebereich des Skalenfaktors für Konstanten des L-Typs: 0 bis 28

3. Erweiterte mnemonische Operationsschlüssel für die Instruktionen BCR und BC

Erweiterter Operationsschlüssel	Bedeutung	Maschineninstruktion bzw. generierter erster Operand
BR	Unbedingte Verzw.	BCR 15, R_2
NOPR	Keine Operation	BCR 0, R_2
B	Unbedingte Verzw.	BC 15, s
NOP	Keine Operation	BC 0, s

Die nachfolgenden Operationsschlüssel werden hauptsächlich nach Vergleichsinstruktionen verwendet

BH | BHR	Verzw. bei hoch	2
BL | BLR	Verzw. bei niedrig	4
BE | BER	Verzw. bei gleich	8
BNH | BNHR	Verzw. bei nich hoch	13
BNL | BNLR	Verzw. bei nicht niedrig	11
BNE | BNER	Verzw. bei nicht gleich	7

Englische Abkürzungen: H High
L Low
E Equal
N Not

Erweiterter Operationsschlüssel	Bedeutung	Generierter erster Operand
Die nachfolgenden Operationsschlüssel werden hauptsächlich nach arithmetischen Instruktionen verwendet		
BO | BOR	Verzw. bei Überlauf	1
BP | BPR	Verzw. bei positiv	2
BM | BMR	Verzw. bei negativ	4
BZ | BZR	Verzw. bei Null	8
BNO | BNOR	Verzw. bei keinem Überlauf	14
BNP | BNPR	Verzw. bei nicht positiv	13
BNM | BNMR	Verzw. bei nicht negativ	11
BNZ | BNZR	Verzw. bei nicht Null	7
Die nachfolgenden Operationsschlüssel werden hauptsächlich nach der Instruktion TM verwendet		
BO | BOR	Verzw. bei Einsen	1
BM | BMR	Verzw. bei Einsen u. Nullen	4
BZ | BZR	Verzw. bei Nullen	8
BNO | BNOR	Verzw., wenn keine Einsen	14
BNM | BNMR	Verzw., wenn nicht Einsen und Nullen	11
BNZ | BNZR	Verzw., wenn keine Nullen	7

Englische Abkürzungen: O Overflow bzw. Ones
P Positive
M Minus bzw. Mixed
Z Zero bzw. Zeros
N Not

Anmerkungen:

1) Bei Verwendung erweiterter mnemonischer Operationsschlüssel darf nur ein Operand geschrieben werden; dieser Operand wird zum zweiten Operanden der übersetzten Instruktion.
2) Durch s wird eine Hauptspeicheradresse spezifiziert.

4. Vereinbarungen

Name	OP	Operanden	Bemerkung
[symbol]	START	absolute Adresse	leer im DOS
[symbol]	COM	leer	
[symbol]	CSECT	leer	
symbol	DSECT	leer	
leer	ENTRY	$r_1, \ldots$	
leer	EXTRN	$r_1, \ldots$	
leer	USING	$v, r_1, \ldots$	
leer	DROP	$r_1, \ldots$	
[symbol]	ORG	[verschieblicher Ausdruck]	leer im DOS
[symbol]	LTORG	leer	
leer	COPY	name	
[symbol]	DC	[wiederholungsfaktor]typ [L länge] [S skalenfaktor] [E Exponent] [konstante]$\cdots$	
[symbol]	DS	wie bei DC	
[symbol]	CCW	befehlsschlüssel, datenadresse, anzeiger, zähler	
symbol	EQU	ausdruck	
leer	CNOP	b, w	
leer	END	[verschiebliche adresse]	
leer	ICTL	anfangsspalte, endspalte, fortsetzungsspalte	
[name]	TITLE	'zeichen'	
leer	PUNCH	'zeichen'	
leer	REPRO	leer	
leer	EJECT	leer	
leer	SPACE	[anzahl der Leerzeilen]	
leer	ISEQ	[linke spalte, rechte spalte]	
leer	PRINT	mindestens 1 Operand aus den folgenden Operandenpaaren: ON|OFF; GEN|NOGEN; DATA| NODATA	
symbol	OPSYN	Operationsschlüssel	nur im OS
operation	OPSYN	leer	nur im OS
leer	POP	PRINT| USING	nur im OS
leer	PUSH	PRINT| USING	
leer	WXTRN	verschiebliches symbol	

Anhang K: Übersicht über das Betriebssystem DOS/VS

1. Die normale Prozedur für das einleitende Programmladen (IPL-Prozedur)

1) Einschalten der Zentraleinheit (ZE)

2) Montieren des Plattenstapels, der als Systemresidenz SYSRES dient.

   Wenn gesonderte Arbeitsplattenstapel verwendet werden, so sind diese ebenfalls zu montieren.

3) Einlegen der folgenden Steuerkarten in den Kartenleser SYSRDR:

   DEL (freigestellt)
   ADD (freigestellt)
   SET (erforderlich)
   CAT (freigestellt)
   DPD (erforderlich)

4) Einstellen der Adresse von SYSRES mit Hilfe der Drehschalter der Ladeeinheit. Bei den Modellen 115 und 125, eintasten der Ladeeinheit-Adresse, gefolgt vom Buchstaben C.

5) Nach dem Aufleuchten der Betriebsanzeigelampen an den Plattenspeichergeräten ist die Ladetaste zu drücken. Bei den Modellen 115 und 125 ist die Eingabetaste zu drükken.

6) Wenn das Kartengerät SYSRDR nach Durchführung der Ladeoperation in den betriebsbereiten Zustand versetzt wird, verursacht die EA-Unterbrechung von SYSRDR, daß die ersten Karten gelesen werden und daß anschließend die Verarbeitung beginnt.

   Wenn SYSRDR vor der Vollendung der Ladeoperation betriebsbereit ist, ist die Unterbrechungstaste zu drücken, damit die Verarbeitung beginnen kann.

7) Über die Konsole können die Befehle DEL, ADD, SET, CAT und DPD ebenfalls eingegeben werden, natürlich dann an Stelle von SYSRDR.

Anmerkungen:

a) DEL- und ADD-Befehle können vermischt eingegeben werden, aber im allgemeinen sollten die DEL-Befehle zuerst kommen. Beide Befehle können nur zur Zeit des einleitenden Programmladens benutzt werden.

b) Das Steuerstatement DEL besitzt die Form:

```
DEL X'kgg'
```

   Es bewirkt, daß das angegebene Gerät mit der Kanal-Geräte-Adresse kgg in der Tabelle der verfügbaren Geräte gestrichen wird.

c) Das Steuerstatement ADD besitzt die Form:

```
ADD X'kee',typ
```

   Unter typ können die nachstehenden Angaben erscheinen:

2501	
2520B1	
2520B2	
2520B3	
2540R	(Kartenleser)
2540P	(Kartenstanzer)
usw.	

Außer den aufgeführten Typangaben sind weitere Typangaben zulässig.

Unter kee ist die Kanal-Geräte-Adresse anzugeben.

Das ADD-Statement bewirkt, daß das aufgeführte Gerät in die Tabelle der verfügbaren Geräte aufgenommen wird, die während der Generierung des Betriebssystemes für den Supervisor angelegt wurde.

d) Das Steuerstatement SET ist in einer der beiden folgenden Formen anzuwenden:

```
SET [DATE=mm/tt/jj,CLOCK=hh/mm/ss]  [,ZONE={EAST}/hh/mm]
                                           {WEST}
SET [DATE=tt/mm/jj,CLOCK=hh/mm/ss]  [,ZONE={EAST}/hh/mm]
                                           {WEST}
```

Welche dieser beiden Formen anzuwenden ist, hängt von der Option ab, die zur Systemgenerierungszeit ausgewählt wurde. Wird ZONE weggelassen, so wird die Greenwicher mittlere Zeit angenommen. Fehlen DATE und CLOCK, so wird die Zeit von TOD angenommen. Ist der TOD-Zeitgeber nicht gesetzt, so ist die Angabe von DATE und CLOCK erforderlich.

e) Das CAT-Steuerstatement weist die Form

```
CAT UNIT=X'kgg'
```

auf und wird zur Neuzuweisung von SYSCAT benutzt. SYSCAT kann nur zur IPL-Zeit neu zugewiesen werden.

f) Das DPD-Steuerstatement besitzt die Form

```
DPD [VOLID=sss sss]
```

Unter ssssss wird die Seriennummer des Datenträgers angegeben, der die Seitendatei enthalten soll. Fehlt VOLID, so erfolgt keine Prüfung auf den richtigen Datenträger.

g) Bei den Druckern der Typen IBM 5203, IBM 3203 und IBM 3211 ist normalerweise der Formularsteuerpuffer und der UCB zu laden. Hierzu dient der nachstehende Job:

```
// JOB    jobname
// EXEC   SYSBUFLD
   FCB SYSxxx [,phasenname [,NULMSG]]
   UCB SYSxxx,phasenname [,FOLD] [,NOCHK] [,NULMSG]
/*
/&
```

Fehlt die Angabe von phasenname auf der FCB-Karte, so muß das FCB-Abbild über SYSIPT gelesen werden. Die zu benutzenden Phasennamen sind bei der Installation festzulegen.

Bei der IBM 1403 ist der UCS-Befehl zum Laden des Puffers zu benutzen.

2. Jobsteuerbefehle

1) CANCEL
Der Job wird abgebrochen. Bei Wirksamkeit von OPTION DUMP wird ein Arbeitsspeicherabzug auf SYSLST gedruckt.

2) DVCDN X'kgg'
Das spezifizierte Gerät wird als nicht verfügbar angesehen.

3) DVCUP X'kgg'
Das sepzifizierte Gerät wird als verfügbar angesehen. Durch diesen Befehl erfolgen jedoch keine Zuweisungen zu dem betreffenden Gerät.

4) LOG
Die Jobsteuerstatements werden auf SYSLOG gelistet.

5) NOLOG
Beendigung der Auflistung der Jobsteuerstatements auf SYSLOG.

6) PAUSE
Unterbindung der Verarbeitung. Zur Wiederaufnahme ist END einzugeben.

7) ROD
Bewirkt die Aufdatierung der statistischen Datensätze auf SYSREC. Dieser Befehl muß bei Systemen ohne Software-Aufzeichnung unterbleiben.

8) SET [DATE=mm/tt/jj|DATE=tt/mm/jj]
[UPSI=xxxxxxxx]
[LINECT=nn]
[RCLST=n]
[RCPCH=n]
[RF=option]
[HC=option]
[SVA=(nK,nK)]
[SDL=CREATE]

Die Operanden können in einer beliebigen Reihenfolge niedergeschrieben werden. Durch Kommas sind die Operanden voneinander zu trennen. Die Form für das Datum wird vom Systemprogrammierer vorgegeben. LINECT bezieht sich auf die Zeilenzahl pro Seite (zwischen 30 und 99). RECLIST führt zur Ausgabe einer Warnung, wenn SYSLST einem Plattenspeicher zugeordnet ist und nur noch Plätze für n Sätze ($100 \leq n \leq 65535$) verbleibt. RCPCH bewirkt das gleiche, wenn SYSPCH einem Plattenspeicher zugeordnet ist. RF=CREATE besagt, daß eine Berichtsdatei, IJSYSRC auf SYSREC initialisiert werden muß. RF=YES besagt, daß eine vorhandene Berichtsdatei benutzt werden kann. HC=CREATE bedeutet die Initialisierung einer Hardcopy-Datei, IJSYSCN auf SYSREC. HC=YES sagt aus, daß eine existierende benutzt werden kann. HC=NO besagt, daß keine Hardcopy angefertigt werden soll. Standard ist HC=YES. Durch SVA wird der Umfang des virtuellen Gebietes für das System festgelegt, der zweite Wert gehört zum GETVIS-Gebiet. Dieser Wert überschreibt den vom Systemprogrammierer festgesetzten Wert. SDL bewirkt das Lesen von Phasennamen über SYSIPT; diese werden zum Aufbau eines neuen Systemverzeichnisses im SVA-Gebiet benutzt. RF, HC, SVA und SDL können nur hinter IPL und vor dem ersten Job gegeben werden.

9) UCS SYSxxx,phasenname[,FOLD]
[,BLOCK][,NULMSG]

Durch SYSxxx ist der Drucker mit Universaltypensatz identifiziert, dessen Puffer geladen werden soll. Unter phasenname ist die 240 Bytes lange Pufferladung zu verstehen; diese ist um eine Kennzeichnung ergänzt. Wird FOLD angegeben, so werden die Kleinbuchstaben in Großbuchstaben umgesetzt. Bei Angabe von BLOCK werden Datenfehler ignoriert. Ist NULMSG aufgeführt, so werden keine Identifizierungsnachrichten auf SYSxxx gedruckt.

3. Jobsteuerbefehle und Jobsteuerstatements

Die folgenden Aussagen können entweder vom Operateur als Jobsteuerbefehle oder, wenn ihnen die Zeichenfolge //b [1] vorausgeht, als Jobsteuerstatements benutzt werden.

---

1) b bedeutet hier das Leerzeichen.

1) `ASSGN SYSxxx,UA[,TEMP|,PERM]`
`ASSGN SYSxxx,IGN[,TEMP|,PERM]`
`ASSGN SYSxxx,X'kgg'[,X'ss'|,ALT][,TEMP|,PERM][,VOL=ssssss[,SHR]]`
`ASSGN SYSxxx,zu[,X'ss'|,ALT][,TEMP|,PERM][,VOL=ssssss[,SHR]]`

Die Angabe von ALT ist nur bei Magnetbandgeräten erlaubt. Durch VOL ist die Seriennummer des Plattenstapels ins Spiel zu bringen. VOL wie auch SHR sind nur bei Plattenspeichern erlaubt; X'ss' gilt nur für Magnetbänder (X'C8' → 800 bpi, X'CØ' → 1600 bpi, X'DØ' → 6250 bpi). Für zu können die folgenden Angaben erfolgen:

(X'kgg',X'kgg',···)
SYSyyy
gerätetyp (z.B. 3420T9)
geräteklasse (z.B. READER, PRINTER, PUNCH, TAPE oder DISK)

2) `CLOSE SYSnnn[,X'kgg'][,X'ss']`
`CLOSE SYSnnn[,UA]`
`CLOSE SYSnnn[,IGN]`
`CLOSE SYSnnn[,ALT]`

Das angegebene EA-Gerät wird abgeschlossen. Wenn X'kgg' spezifiziert wird, wird SYSnnn neu zugeordnet, d.h. es läuft der gleiche Vorgang ab wie bei einem ASSGN-Befehl. Bei Verwendung von UA oder IGN wird der gleiche Effekt wie beim ASSGN-Befehl erreicht. Bei Angabe von ALT wird das Gerät abgeschlossen und ein Wechselbandgerät neu eröffnet. IGN ist für SYSIN, SYSIPT und SYSRDR ungültig.

3) `EXEC [[PGM=]programmname[,REAL][SIZE=umfang]]`
`EXEC [PROC=prozedurname][,OV]`

Der Programmname kann entfallen, wenn die Phase gerade in den temporären Teil der Bibliothek ausführbarer Programme hineingestellt wurde, und zwar als Ergebnis eines Binderlaufes. REAL sagt aus, daß das Programm im Realspeicher auszuführen ist; dabei nennt SIZE den Umfang desselben. Fehlt REAL, so nennt SIZE den Umfang des dem Programm zugewiesenen Speicherplatzes, der für GETVIS verfügbar ist. PROC nennt den Namen einer Prozedur; OV zeigt an, daß Statements folgen, die zu Überschreitungen führen. Ein auf eine Prozedur bezogenes Statement ist für SYSLOG erlaubt, nicht jedoch für SYSRDR.

4) `LISTIO einheiten`
Die EA-Zuweisungen für die Geräte werden aufgelistet. Gültige Operanden sind:

SYS	für logische Systemeinheiten
PROG	für logische Einheiten des Programmierers
BG, F1, F2, F3, F4	für Einheiten der betreffenden Partition
ALL	für alle Einheiten aller Partitions
SYSxxx	für alle SYSxxx zugewiesenen Einheiten
UNITS	für alle Geräte
DOWN	für alle nicht eingesetzten Geräte
X'kgg'	für das angegebene Gerät

5) MTC op,SYSxxx[,nn]
MTC op,X'kgg'[,nn]

Die Angabe von X'kgg' ist nur bei Befehlen erlaubt. Unter op kann eine der folgenden Angaben gemacht werden:

BSF, BSR, ERG, FSF, FSR, RUN, REW, WTM, DSE

Unter nn ist anzugeben, wieviel Mal die betreffende Operation ausgeführt werden soll. Standard ist 01.

6) OVEND
Auf ein EXEC-Statement mit dem OV-Operanden folgend, bezeichnet dieser Befehl bzw. dieses Statement das Ende der zum Überschreiben heranzuziehenden Befehle bzw. Statements.

7) RESET SYS
RESET PROG
RESET ALL
RESET SYSxxx

Zeitweilige Gerätezuweisungen für SYS (logische Systemeinheiten), PROG (vom Programmierer vergebene Einheiten), ALL (alle Einheiten) oder für SYSxxx werden aufgehoben.

4. Jobsteuerstatements

1) // DATE mm/tt/jj
// DATE tt/mm/jj

Das Datum wird im Verständigungsbereich gespeichert und wird am Jobende gelöscht.

2) // DLBL [dateiname],['dateikennzeichnung'],[datum],[schlüssel]

Hinter dem letzten Operanden kann das Komma bzw. können die Kommas fehlen. Unter dateiname kann ein bis zu siebenstelliger Name angegeben werden. Standards sind für

dateikennzeichnung	der Dateiname
datum	eine siebentägige Aufbewahrungsfrist

Für schlüssel wird SD angenommen; andere mögliche Angaben sind DA, ISC, ISE und VSAM.

Das Datum kann als Aufbewahrungszeitraum oder als Erstellungsdatum (Eingabe) oder als Verfallsdatum (Ausgabe) in der Form jj/ttt angegeben werden.

3) // EXTENT [SYSxxx],[seriennummer],[typ],[reihenfolge],
[relative-spur],[anzahl-der-spuren],
[zylinderspur]

Kommas hinter dem letzten Operanden können weggelassen werden. Fehlt die Angabe von SYSxxx, so wird diegleiche Angabe unterstellt, die im letzten EXTENT-Statement auftrat; für das erste Statement wird sie aus dem Programm entnommen. Ist die Angabe von seriennummer weggelassen, so wird diegleiche wie beim letzten EXTENT-Statement benutzt. Unter typ ist zu schreiben:

1	bei Dateibereichen
2	bei Überlaufbereichen von indexsequentiell organisierten Dateien
4	bei Inhaltsverzeichnisbereichen von indexsequentiell organisierten Dateien
8	bei Datenbereichen auf aufgeteilten Zylindern

Unter reihenfolge ist die fortlaufenende Bereichsnummer anzugeben; sie beginnt bei 0, ausgenommen bei indexsequentiell organisierten Dateien ohne Hauptindex, bei denen sie bei 1 beginnt. Unter relative-spur ist eine ein- bis fünfstellige Zahl zu nennen, die die erste Spur des Bereiches identifiziert. Für anzahl-der-spuren ist eine ein- bis fünfstellige Zahl anzugeben. Durch zylinderspur wird die Nummer des Kopfes genannt, der die Schlußspur einer Datei in den Zylindern bestreicht (unter verschiedenen Dateien aufgeteilte Zylinder). Die letzten drei Operanden werden nur für Ausgabedateien benötigt.

4) `// JOB jobname`

Der Jobname kann aus 1 bis 8 alphanumerischen Zeichen bestehen. Das UPSI-Byte, der Bereich für den Informationsaustausch und die zeitweiligen Gerätezuordnungen (TEMP) werden gelöscht.

5) `// BLBTYP TAPE [nn]`
`// LBLTYP NSD(nn)`

Bei Magnetbändern wird die Angabe von nn ignoriert. Bei Plattenspeichern bedeutet nn die Anzahl der nicht starr fortlaufend organisierten Dateibereiche mit zu prüfenden Kennsätzen. NSD sollte nur dann verwendet werden, wenn sowohl Band- als auch Plattendateien vorhanden sind. Die Angabe von nn hinter TAPE wird ignoriert.

Dieses Statement gilt nicht für VSAM-organisierte Dateien.

6) `// OPTION option1,option2,...`

Es gibt folgende Optionen:

a) LOG	Die Steuerstatements werden auf SYSLST gelistet
b) DUMP	Es erfolgt ein Hauptspeicherauszug, wenn es zu einem irregulären Programmabbruch kommt
c) LINK	Die Ausgabe des Assemblierers bzw. der Kompilierer wird auf die Einheit SYSLNK aufgezeichnet, wodurch es möglich wird, daß ihn der Linkage Editor verarbeiten kann
d) DECK	Die Ausgabe des Assemblierers bzw. der Kompilierer wird gestanzt. DECK und LINK unterdrücken sich gegenseitig, d.h. wenn DECK angegeben wird, wird gleichzeitig auch NOLINK angenommen und umgekehrt.
e) LIST	Das Quellenprogramm wird über SYSPRINT aufgelistet
f) LISTX	Die Kompilierer von COBOL und PL/1 geben das Objektprogramm unter Benutzung der AL aus (sie geben eine sogenannte Codeliste aus). Das gleiche gilt für die FORTRAN-Objekt-Moduln
g) SYM	Bei gleichzeitiger Angabe von NODECK wird vom Assemblierer die Symboltabelle über SYSLST ausgegeben; bei Angabe von DECK wird die Symboltabelle über SYSPCH ausgegeben. Diese Option gilt auch für den PL/1-Kompilierer. Der COBOL-Kompilierer gibt die Übersicht (Map) über den Datenteil des Programmes über SYSLST bzw. SYSPCH aus
h) XREF	Der Assemblierer gibt die Zuordnungstabelle (Cross Reference List) über SYSLST aus; gilt auch für den PL/1-Kompilierer
i) ERRS	Die Kompilierer von COBOL und FORTRAN fassen alle Fehler auf SYSLST zusammen; das gleiche gilt auch für den PL/1-Kompilierer

j) 48C	Der PL/1-Kompilierer nimmt an, daß das Quellenprogramm mit dem 48-Zeichen-Satz geschrieben ist
k) 60C	Der PL/1-Kompilierer nimmt an, daß das Quellenprogramm mit dem 60-Zeichen-Satz geschrieben ist
l) CATAL	Das Objektprogramm wird in der Bibliothek für ausführbare Programme katalogisiert. CATAL schließt die Angabe von LINK ein. Das Katalogisieren setzt ein, wenn die Zeichenfolge /& gelesen ist.
m)EDECK	Durch EDECK werden alle gültigen Makrodefinitionen im aufbereiteten Format auf SYSPCH gestanzt. Es wird die Form benutzt, die notwendig ist, um ein Katalogisieren in der E-Quellenstatement-Bibliothek durchführen zu können
n) ALIGN	Ausrichtung der Daten auf geeignete Begrenzungen
o) ACANCEL	Verursachung des Jobabbruches bei nicht erfolgreicher Zuweisung; NOACANCEL erlaubt Eingriff des Bedieners
p) STDLABEL	Kennsätze werden auf die Standardkennsatzspuren von SYSRES geschrieben
q) PARSTD	Kennsätze werden auf die Standardkennsatzspuren der Partitions geschrieben
r) USRLABEL	(Standard) Kennsätze werden auf Kennsatzspuren des Benutzers geschrieben. Nur der Systemprogrammierer kann Kennsätze auf andere Spuren bringen
s) SYSPARM='kette'	Festlegung einer Zeichenkette mit 0 bis 8 Zeichen, die vom Assemblierer benutzt werden kann

Anmerkung: Den Optionen LOG, DUMP, LINK, DECK, EDECK, ALIGN, LIST, LISTX, SYM, XREF, ERRS und ACANCEL kann die Silbe NO vorangestellt werden. Dadurch wird angezeigt, daß die betreffenden Optionen außer Kraft gesetzt werden sollen.

7) `// TLBL dateiname,['dateikennzeichnung'],[datum],`
`[seriennummer],[laufnummer-des-datenträgers],`
`[laufnummer-der-datei],[generation],[version]`

Kommas, die auf den letzten Operanden folgen, können weggelassen werden. Der Dateiname umfaßt 1 bis 7 Zeichen. Standards sind für die Ausgabe:

dateikennzeichnung	Dateiname
datum	Es wird eine Verfallzeit von 0 Tagen angenommen
seriennummer	Laufende Datenträgernummer des ersten Datenträgers der Datei
laufnummer-des-datenträgers	0001
laufnummer-der-datei	0001
generation	0001
version	0001

Bei Eingabedateien hat das Datum die Form jj/ttt und bedeutet das Erstellungsdatum. Bei Ausgabedateien ist das Verfalldatum in der Form jj/ttt anzugeben; man kann auch die Verfallszeit in der Form tttt spezifizieren.

8) `// UPSI xxxxxxxx`
Jedes x bezieht sich auf eine Bitstelle des UPSI-Bytes; x=1 bewirkt, daß sie auf 1 gesetzt wird, x=0, daß sie auf 0 gesetzt wird; x=X läßt die Bitstelle ungeändert. Werden weniger als acht Zeichen notiert, so wird nach rechts mit X aufgefüllt.

9) `// ZONE EAST/hh/mm`
`// ZONE WEST/hh/mm`

Durch dieses Statement wird gesagt, ob die lokale Zeit eine Zeit ist, die zu einer Zone östlich oder westlich von Greenwich gehört

10) `* kommentar`
Der Inhalt der Spalten 1 bis 72 wird auf SYSLOG gedruckt. Spalte 2 muß leer bleiben

11) `/+ [kommentar]`
Durch dieses Statement wird das Ende einer katalogisierten Prozedur markiert. Die Spalte 3 muß leer bleiben

12) `/& [kommentar]`
Durch dieses Statement wird das Ende eines Jobs gekennzeichnet. Die Spalte 3 muß leer bleiben

13) `/* [kommentar]`
Durch dieses Statement wird das Datenende bei SYSIPT oder bei SYSRDR signalisiert. Die Spalte 3 muß leer bleiben. Das logische IOCS akzeptiert/* als Dateiende für jede SYS-Einheit.

Anhang L: Übersicht über das Betriebssystem OS/VS

1. Befehle des Maschinenbedieners (Operateurs)

```
1) { CANCEL } { jobname [,DUMP  ,ALL]                    }
   { C      } { jobname [,IN=eingabeklasse]              }
             { jobname [,IN=HOLD]                       }
             { jobname [,OUT=ausgabeklasse]             }
             { jobname [,OUT=HOLD]                      }
             { [prozedurname,]bezeichnung               }
             {  kgg                                     }
             {  gerätetyp                               }
```

CANCEL führt zur Aufhebung eines Jobs oder zur Aufhebung der Arbeit eines Gerätes bzw. eines Gerätetyps. DUMP bewirkt die Ausgabe eines Speicherabzuges über SYSABEND oder SYSUDUMP. Durch ALL ist festgelegt, daß alle Warteschlangen nach dem Job durchsucht werden sollen, während HOLD besagt, daß nur die Festhaltewarteschlange zur Durchsuchung herangezogen werden soll. Wenn weder IN noch OUT noch DUMP spezifiziert sind, wird ALL angenommen. Die letzten drei Befehle führen zur Aufhebung von Arbeiten, die durch einen START-Befehl begonnen wurden. Das anzusprechende Gerät muß dem des START-Befehles entsprechen.

```
2) { DISPLAY } { R                                                          }
   { D       } { A                                                          }
             { Q[=klasse]                                                   }
             { N[=klasse]                                                   }
             { jobname[,HOLD]                                               }
             { U[,TAPE ][,ONLINE ][,kgg][,nnn]                              }
             {  ,DASD  ][,OFFLINE ]                                         }
             {  ,UR    ]                                                    }
             { SQA                                                          }
```

Die erste Befehlsform (R) dient zur Anzeige von ausstehenden Maßnahmen des Operateurs und von Antworten.

Die zweite Befehlsform (A) bewirkt die Anzeige aller aktiven Tasks.

Die dritte Befehlsform (Q) zeigt die Anzahl der Jobs an, die in allen Klassen vorhanden sind. Wird eine Klasse (SOUT, HOLD oder A bis O) niedergeschrieben, so werden nur die Jobs angezeigt, die zur spezifizierten Klasse gehören. Man kann auch den Status mehrerer Klassen anzeigen lassen, doch dann sind die Klassen zu einer von Klammern umgebenen Liste zusammenzufassen und durch Kommas voneinander zu trennen.

Die vierte Befehlsform (N) bewirkt die Anzeige der Namen der Jobs in allen Klassen oder in den spezifizierten.

Die vierte Befehlsform mit der durch jobname erfolgenden Bezugnahme auf einen Job führt zur Anzeige des Status dieses Jobs. Wird HOLD spezifiziert, so werden die Festhaltewarteschlangen der Eingabe und Ausgabe durchsucht.

Die fünfte Befehlsform (U) führt zur Anzeige des Status aller EA-Einheiten oder zu der spezifizierten. Wird kgg und nnn niedergeschrieben, so beginnt die Statusanzeige beim Gerät mit der Adresse kgg und umfaßt nnn Geräte. Wird kgg weggelassen, so wird dafür standardmäßig 000 angenommen; das Komma vor nnn ist dann erforderlich. Wird nnn weggelassen, so wird dafür standardmäßig der Wert 100 eingesetzt.

Die letzte Form schließlich zeigt die Begrenzung der Systemwarteschlange an (beim System OS/VS1).

3) { HALT } EOD
   { Z   }

Dieser Befehl vor dem Ausschalten des Systems sichert die Rettung der Informationen über Hardwareprobleme.

4) { LOG } 'text'
   { L   }

Der angegebene Operand wird in das Systemlogbuch gestellt; dabei entfallen die begrenzenden Hochkommas.

5) { MODIFY } { [prozedurname.] bezeichnung [,CLASS=ausgabeklassen][,PAUSE={FORMS}]   }
   { F   } { [prozedurname.] bezeichnung [,CLASS=jobklasse]   }
   { {kgg   }{,JOBCLASS=jobklassen   }
   { {[prozedurname.] bezeichnung   }{,OUTCLASS=ausgabeklassen   }
   { {   }{,JOBCLASS=jobklassen,OUTCLASS=ausgabeklassen   }

a) Die erste Befehlsform wird für den Ausgabeschreiber benutzt, um die zu verarbeitenden Klassen zu modifizieren, PAUSE=FORMS bewirkt ein Warten auf Formularwechsel, PAUSE=DATASET ein solches bei jeder neuen Datenmenge.

b) Die zweite Befehlsform spricht einen Initiator an; durch ihn können zur Verarbeitung anstehende Jobklassen geändert werden.

c) Die dritte Befehlsform spricht DSO-Schreiber an; mit ihr können die Ausgabeklassen geändert werden, die der Schreiber verarbeiten kann oder die Jobklassen, die den Schreiber benutzen können. Die angegebene Bezeichnung identifiziert den Schreiber.

6) { MONITOR } { JOBNAMES[,T] }
   { MN   } { A   }

Durch diesen Befehl wird jeder gestartete oder beendete Job angezeigt. Bei zusätzlicher Angabe von T erfolgt auch die Zeitmitteilung.

Bei der zweiten Form (A) wird auf dem Bildschirm in periodischen Abständen der Status aller aktiven Jobs angezeigt. Diese Form ist nur dann gültig, wenn mehrere Konsolen unterstützt werden.

7) { MOUNT } kgg,VOL=({NL},seriennummer)[,USE {STORAGE}]
   { M   }   {SL}   {PUBLIC }
                                             {PRIVATE}

Das Gerät mit der Adresse kgg muß durch einen UNLOAD-Befehl oder durch das System (kenntlich gemacht durch eine Nachricht) entladen sein. Der Montierungsbefehl macht eine Platte oder ein Band für alle Jobs verfügbar, die diesen Datenträger benötigen. Der Datenträger wird vom System erste bei Ausführen eines UNLOAD-Befehles wieder freigegeben.

8) { RELEASE } { jobname   }
   { A   } { Q   }
   {   } { Q=klassen }

Der durch jobname bezeichnete Job wird aus der Festhaltewarteschlange freigegeben (1. Form).

Die zweite Befehlsform veranlaßt die Freigabe aller Eingabewarteschlangen von der Festhaltewarteschlange.

Bei der dritten Form werden alle Eingabejobs in den benannten Klassen freigegeben.

9) { REPLY } { kz,'text' }
   { R   } { kz,text   }

Unter kz ist die ein- oder zweistellige Nachrichtennummer zu verstehen, auf die die Antwort des Operateurs erfolgt. Die Verwendung der begrenzenden Hochkommas ist nach dem einleitenden Programmladen freigestellt.

10) { RESET } jobname [,PRTY=n] [,CLASS=c] [,OUT=s]
    { R   }

Mindestens einer der Operanden PRTY und CLASS ist erforderlich. Der Job wird bei Benutzung von PRTY auf die Priorität n mit $0 \leq n \leq 13$ zurückgesetzt. Fehlt OUT, so wird der Job in die Jobklasse c (A bis O) gebracht. Bei Anwesenheit von OUT wird die Ausgabeklasse s für diesen Job auf die Ausgabeklasse c (A bis Z, 0 bis 9) gesetzt.

11) { SET } DATE=jj.ttt [,CLOCK=hh.mm.ss]
    { T   }

Dieser Befehl ändert nach dem einleitenden Programmladen Datum und Zeit.

12) { START } proz [.id], [kgg / gerätetyp] , [datentr] , [par] [[schlüsselwort=option] ...]
    { S   }

Die normalen Prozedurnamen sind RDR, WTR, INIT und DSO. Für INIT und DSO ist unter id die Bezeichnung Pnn einzusetzen (im VS1). Bei den Prozeduren RDR und WTR darf unter id keine Bezeichnung benutzt werden, die für INIT benutzt wird.

Im VS1 kann ALL zusammen mit INIT benutzt werden; damit können die Initiatoren aller Partitionen gestartet werden. Als Parameter können unter par folgende Angaben erscheinen:

a) RDR
   Name des ersten zu verarbeitenden Jobs,

b) WTR
   Zu verarbeitende Ausgabeklassen. Dabei sind die Formen (A,B,C) und ABC einander gleichwertig,

c) DSO
   JOBCLASS=c und/oder OUTCLASS=c in beliebiger Reihenfolge,

d) INIT

   jobklasse oder (jobklasse,RESV=nn)

   Unter nn ist die Anzahl (standardmäßig 15) der zu reservierenden SWADS-Sätze zu nennen. Die Reihenfolge von jobklasse und RESV=nn ist beliebig.

Die Schlüsselwörter sind die gleichen, die im DD-Statement für die Prozedur erscheinen dürfen.

13) { STOP } { proz.id }
    { P   } { kgg   }
    {   } { JOBNAMES }

Die unter id zu nennende Bezeichnung entspricht der des START-Befehles. Für Leser und Schreiber kann auch die Adresse kgg angegeben werden. Bei Nennung von JOBNAMES sind alle MONITOR JOBNAMES gemeint.

14) { SWITCH } SMF
    { I   }

Es erfolgt der Wechsel von einer SMF aufzeichnenden Datenmenge zu einer anderen,

15) { UNLOAD } kgg
    { U   }

Damit wird ein Datenträger darauf vorbereitet, daß er so bald als möglich entladen wird, und zwar zum Zeitpunkt des Beginns der nächsten Verarbeitungsfunktion,

16) {VARY / V} kgg, {ONLINE / OFFLINE}
(kgg1,... {ONLINE / OFFLINE}

Die angegebenen Geräte werden so bald als möglich in den ONLINE-Zustand oder in den OFFLINE-Zustand versetzt,

17) {WRITELOG / W} {ausgabeklasse / CLOSE}

Die Systemlogausgabe wird zur spezifizierten Ausgabeklasse geschickt. CLOSE beendet die Logfunktion und wird wie HALT EOD benutzt.

18) {WRITER / WTR} kgg,funktion[,JBN=jobname]

Unter kgg ist die Adresse des Druckers zu verstehen, der vom Schreiber benutzt wird und dessen Ausgabezustand geändert werden soll. Fehlt der Operand JBN, so ist der gerade ausgegebene Job gemeint. Für funktion können folgende Angaben gemacht werden:

a) FSP=nnn
Vorwärtsgehen über nnn (mit $1 \leq nnn \leq 255$) logische Seiten hinweg

b) FSP=DS
Vorwärtsbewegung bis zur nächsten Datenmenge

c) BSP=nnn
Rückwärtsgehen über nnn (mit $1 \leq nnn \leq 100$) logische Seiten hinweg

d) BSP=DS
Rückwärtsbewegung bis zum Beginn der Datenmenge, danach erneuter Druckbeginn

e) BSP=JOB
Rückwärtsgehen bis zum Jobbeginn und danach erneutes Starten des Druckens

f) LSP=n
Zeilentransport über n Zeilen (mit n=1,2,3) hinweg ohne Rücksicht auf die Angabe im Job

g) LSP=C
Aufhebung des letzten Befehles LSP=n

h) HOLD
Aufheben des Jobs unter erneuter Einreihung in die Warteschlange

i) REPEAT=nnn
Es sind zusätzlich nnn (mit $1 \leq nnn \leq 254$) Kopien der Datenmenge anzufertigen

j) REPEAT=(nnn,JOB)
Es sind zusätzlich nnn Kopien des Jobs anzufertigen

19) ROUTE JBM=jobname,HOLD=NO
Freigabe des Jobs, der durch den Befehl WRITE und HOLD festgehalten wurde.

2. Statements der Jobsteuersprache (JCL)

2.1. Das Steuerstatement JOB (JOB-Steuerkarte)

Die Angabe aller Operanden ist freigestellt. Es können jedoch die Abrechnungsinformationen und der Programmierername zu Pflichtoperanden gemacht werden.

Die Einleitung des Statements erfolgt durch

```
//jobname  JOB  ...
```

Nachstehend sind die Operanden aufgeführt und ihre Bedeutung erklärt:

([abnr] [,abrinf])	Abrechnungsinformationen. Bei Weglassung von abruf können die Klammern entfallen
,programmierername	Der Programmiername muß in Hochkommas stehen, wenn er andere Sonderzeichen als den Punkt enthält
ADDRSPC={VIRT / REAL}	
CLASS=jobklasse	Als Standardangabe gilt die Jobklasse A
COND=((code,operator),...)	Die Codes code rangieren zwischen 0 und 4095 einschließlich. Als Operatoren können GT, GE, EQ, LT, LE und NE benutzt werden. Wenn eine Bedingung code operator returncode war, so werden die verbleibenden Jobsteps übergangen. Wird nur eine Bedingung benutzt, so können die äußeren Klammern entfallen
MSGCLASS=sakl	Standardmäßig gilt für Nachrichtenklasse die Systemausgabeklasse A
MSGLEVEL=(x [,y])	Fehlt die Angabe von y, so können die Klammern wegfallen. Ist x=0, so wird nur das Job-Statement ausgegeben. Bei x=1 kommt es zur Ausgabe aller Statements der Jobsteuersprache (JCL) sowie aller in den Prozeduren versteckten Steuerstatements. Ist x=2, so werden nur die normalen Beendigungs- und Zuordnungsnachrichten geschrieben, bei y=1 alle. Standard ist MSGLEVEL=(1,0)
PRTY=n	Für n kann ein numerischer Wert zwischen 0 und 13 notiert werden (Priorität des Jobs). Der Standardwert wird der Leserprozedur des Systems entnommen. PRTY=13 sollte vom Programmierer nicht benutzt werden.
REGION=nK	Durch diesen Operanden wird der Umfang der realen Partition festgelegt. Wird für virtuelle Partitions vernachlässigt. Der Standardwert wird von der Leserprozedur des Systems genommen

`TIME=(min,sek)` — Durch diesen Operanden wird die maximale Jobzeit in Minuten und Sekunden festgelegt. Bei Abwesenheit des zweiten Wertes sek können die Klammern fehlen. TIME=1440 spezifiziert einen endlos laufenden Job.

`TYPRUN=HOLD` — Der Job wird solange in die Festhaltewarteschlange gestellt, bis er vom Operateur zur Ausführung freigegeben wird

2.2. Das Steuerstatement EXEC (EXEC-Steuerkarte)

Der Stepname wird nur dann benötigt, wenn in irgendeinem anderen Steuerstatement eine Bezugnahme auf ihn erfolgen muß.

Das EXEC-Statement beginnt mit

```
// [stepname]  EXEC  ...
```

Nachstehend sind die Operanden zusammengestellt und z.T. ihre Bedeutung erklärt:

```
{PGM=programmname                  }
{PGM=*.sn[.psn].ddname             }
{[PROC=]prozedurname               }
```

Dieser Operand muß als erster erscheinen. Die Benutzung aller anderen Operanden ist freigestellt. Unter sn ist der Stepname und unter psn der Prozedurstepname zu verstehen

`ACCT[.psn]=[abrifn]` — Die Abrechnungsinformationen abrinf werden in diesem Operanden festgehalten

```
ADDRSPC={VIRTUAL}
        {REAL   }
```

Bei Fehlen dieses Operanden wird standardmäßig VIRTUAL angenommen

```
COND[.psn]=(bed,...[,EVEN])
                   [,ONLY]
```

Wenn nur eine Bedingung bed anzugeben ist, können die Klammern wegfallen. Jede Bedingung besitzt die Form

`(code,operator[,stepname[.prozedurstepname]])`

Fehlt die Angabe des Stepnamens, so werden alle zuvor ausgeführten Jobsteps auf ihren Rückkehrcode getestet. EVEN besagt, daß der Step auf jeden Fall ausgeführt werden soll, gleichgültig, ob dabei ein vorangegangener Jobstep anormal endete. Hingegen bedeutet ONLY, daß der Jobstep nur dann zur Ausführung kommt, wenn mindestens ein vorangehender Jobstep anormal endete.

`DPRTY=(m,n)` — Durch diesen Operanden wird die Priorität für den Jobstep festgelegt (gilt nur unter dem OS/VS2)

`PARM[.psn]=wert` — Es können maximal 100 Zeichen spezifiziert werden

`REGION=nK` — Durch diesen Operanden wird der Umfang der realen Partition festgelegt

`TIME[.psn]=(min,sek)` — Durch diesen Operanden, der in gleicher Form wie für das JOB-Statement gilt, kann die Maximallaufzeit für den Jobstep festgelegt werden

2.3. Kommentare (Kommentarkarten)

Ein Kommentar beginnt mit der Zeichenfolge

```
//*
```

auf den Spalten 1 bis 3. Der Rest der Karte kann den Kommentar aufnehmen.

2.4. Leerstatement (Leerkarte)

Enthält ein Statement auf den Spalten 1 bis 3 die Zeichen

```
//b 1)
```

so wird es als Leerstatement angesehen.

2.5. Statement für das Ende einer Datenmenge (Datenmengenendekarte)

Auf den Spalten 1 und 2 steht die Zeichenfolge

```
/*
```

2.6. Das DD-Steuerstatement (DD-Karte)

Die DD-Karte wird eingeleitet durch

```
//[prosstep.] ddname  DD  ...
```

Es gibt keine festen Operanden. Alle nachstehend aufgeführten und zum Teil kommentierten Operanden sind Wahloperanden, d.h. ihre Angabe ist optional.

*	Die Daten folgen im Eingabestrom hinter diesem Statement und gehen bis zu einer Karte mit // oder /* auf den Spalten 1 und 2
DATA	Die Daten folgen im Eingabestrom bis zu einer Karte mit /*
DUMMY	Alle auf die betreffende Datenmenge bezogenen EA-Operationen werden übergangen
AFF=ddname	Durch diesen Operanden wird die gleiche Kanaltrennung wie für das DD-Statement namens ddname durchgeführt; dieses muß den Operanden SEP besitzen
AMP=('AMORG' 'BIFMD=n' 'BUFNI=n' 'BUFSP=n' 'OPTCD={I [L] / L}' 'RECFM={F [B] / V [B]}' 'STRNO=n' 'SYNAD=modulname')	Der Operand AMP gilt nur für VSAM. AMORG ist erforderlich, wenn DUMMY, UNIT oder VOLUME spezifiziert sind. BUFND, BUFNI und BUFSP überschreiben die entsprechenden Angaben im ACB. Für die Schnittstelle zu ISAM gilt der Parameter OPTCD, soweit es sich um Sätze handelt, die zur Löschung markiert sind: IL löscht solche Sätze, I verarbeitet sie, wenn im DCB der Buchstabe L enthalten ist, L entfernt sie nur dann, wenn sie ersetzt werden. Der Parameter RECFM betrifft nur die ISAM-Schnittstelle. Die Operanden STRNO und SYNAD überschreiben die entsprechenden Angaben im ACB bzw. im EXLST.
COPIES=n	Anzahl der Kopien, die von einer zur Ausgabe anstehenden (SYSOUT) Datenmenge angefertigt werden sollen

1) b bedeutet hier das Leerzeichen.

```
DCB=(attribut,···)
DCB=(dsname,attribut,···)
DCB=(* [.sn [.psn]] .ddname,attribut,···)
```

Wird nur ein Attribut angegeben, so können die Klammern weggelassen werden. Die Attribute können im DCB-Operanden in einer beliebigen Reihenfolge angegeben werden. Wird dsname oder ddname niedergeschrieben, so werden die Attribute von der betreffenden Datenmenge kopiert, es sei denn, sie werden von hier aufgeführten Attributen außer Kraft gesetzt. Unter sn wird der Stepname, unter psn der Prozedurstepname verstanden

```
DDNAME=ddname
```

Durch diesen Operanden wird die Definition der Datenmenge solange aufgeschoben, bis das DD-Statement für ddname erreicht ist

```
        [SHR]   [DELETE  ]  [DELETE  ]
        [NEW]   [KEEP    ]  [KEEP    ]
DISP=(  [OLD] , [PASS    ] , [CATLG   ] )
        [MOD]   [CATLG   ]  [UNCATLG ]
        [    ]  [UNCATLG ]  [        ]
```

Kommas, die auf den letzten Parameter folgen, können weggelassen werden

```
DLM=xy
```

Die beiden Zeichen x und y werden an Stelle von /* als Begrenzung für eine Datenmenge angesehen. Voraussetzung ist, daß dieser Operand nur zusammen mit dem Operanden * oder DATA benutzt wird. Bei Benutzung von Sonderzeichen muß xy in Hochkommas gesetzt werden

```
{DSNAME}   {[&&] name [(gliedname)   ]}
{DSN   }=  {          [(gebietsname)]}
{      }   {name [(generationsnummer)]}
{      }   {* [.sn [.psn]] .ddname    }
```

```
FCB=(abbild [,ALIGN ] )
            [,VERIFY]
```

Dieser Operand gilt nur für eine Datenmenge, die über ein Gerät der Typen IBM 5203, IBM 3203 oder IBM 3211 geschrieben wird

```
HOLD={YES}
     {NO }
```

Dieser Operand kann dazu benutzt werden, um die SYSOUT-Ausgabe in die Festhaltewarteschlange zu stellen

```
LABEL=([dateinummer],[kennsatztyp],
       [PASSWORD] , [IN ]
       [NOPWREAD] , [OUT] ,
       [EXPDT=jjttt ]
       [RETPD=nnnn  ])
```

Unter kennsatztyp kann SL, SUL, NSL, NL, BLP oder LTM aufgeführt werden. BLP heißt Übergehen der Kennsatzprüfung, LTM verweist auf eine führende Bandmarke. Standardmäßig wird SL angenommen

```
OUTLIM=n
```

Maximalzahl der logischen Sätze, die geschrieben werden können, bevor der Step anormal beendet wird

```
SEP=(ddname,···)
```

Durch diesen Operanden wird angefordert, daß Kanaltrennung von den Kanälen gefordert wird, über die die Datenmengen ddname,··· laufen

```
SPACE=(ASTR,(prim,adresse,
        [verzeichnis]))
        [index      ]
```

```
SPACE=( {TRK      },
        {CYL      },
        {blocklänge}
        (prim,[sekundär],
         [verzeichnis]),
         [index     ]
        [RLSE],
        [CONTIG]
        [MXIG  ],
        [ALX   ]
        [ROUND])
```

```
SPLIT=n
```

```
SPLIT=(n,CYL,(prim [,sekundär]))
```

```
SPLIT=prozent
```

```
SPLIT=(prozent,blocklänge,
       (prim [,sekundär]))
```

```
SUBALLOC=( {TRK      },
           {CYL      },
           {blocklänge}
           (prim[,sekundär]
            [,verzeichnis]),
           [sn[.psn]] .ddname
```

```
SYSOUT=(klasse,[prona],[form])
```

Die Klammern können entfallen, wenn nur klasse als Parameter auftritt. Unter prona ist der Programmname und unter form die Formularnummer zu verstehen

```
USC=(zeichenmenge[,FOLD][,VERIFY])
```

Wenn nur der erste Parameter benötigt wird, können die Klammern entfallen. Wenn dieser Operand für eine SYSOUT zugeordnete IBM 1403 verwendet wird, wird UCS ignoriert

```
UNIT=AFF=ddname
```

Es wird die gleiche EA-Einheit angefordert, wie für die unter ddname genannte Datenmenge

```
UNIT=( [einheitenadresse] [,P]
       [gerätetyp       ] [,n]
       [gruppenname     ]
       [,DEFER]
       [,SEP=(ddname,···)])
```

P zeigt an, daß jedem Datenträger ein Gerät zugeordnet werden soll, anderenfalls werden n Geräte zugeordnet. Standardwert ist n=1

```
{VOLUME}=([PRIVATE] [,RETAIN]
{VOL   }  [,datenträgernummer]
          [,datenträgerzähler])
          [SER=(seriennummer,···)]
          [REF=dsname            ]
          [REF=* [.sn [.psn] .ddname]
```

3. Die Parameter des EXEC-Steuerstatements für katalogisierte Prozeduren

In den nachstehenden Übersichten sind diejenigen Parameter unterstrichen, die standardmäßig angenommen werden, wenn ein Parameter der betreffenden Gruppe fehlt.

3.1. PL/1

Parameter (Optionen)	*Abkürzung*	*Bedeutung*
AGGREGATE | NOAGGREGATE	AG | NAG	Der Kompilierer gibt die Länge aller Bereiche und aller Hauptstrukturen aus
ATTRIBUTES | NOATTRIBUTES	N | NA	Drucken der Attribute aller Bezeichnungen
CHARSET([48|6Ø] [EBCDIC] [BCD])	CS([48|6Ø] [EB] [B])	Zeichenvorrat des Quellenmoduls
COMPILE | NOCOMPILE [(W|E|S)]	C | NC [(W|E|S)]	Die Angabe von NC ohne nachfolgenden Buchstaben bewirkt den Abbruch der Übersetzung, wenn während der Syntaxprüfung irgend ein Fehler entdeckt wird; NC mit einem Folgebuchstaben führt zum Stop der Übersetzung, wenn ein Fehler der Grade W, E bzw. S festgestellt wird
CONTROL('paßwort')	-	Paßwort, das die Benutzung von Optionen erlaubt, die vom Systemprogrammierer nicht vorgesehen sind
COUNT | NOCOUNT	CT | NCT	Der Kompilierer erzeugt einen Maschinencode, mit dessen Hilfe es möglich ist festzustellen, wieviel jedes Statement während der Programmausführung angesprochen wurde
DECK | NODECK	D | ND	Ausgabe des Objektmoduls über SYSPCH
DUMP | NODUMP	DU | NDU	Der Kompilierer gibt einen Speicherabzug aus, wenn er abnormal endet
ESD | NOESD	-	Der Kompilierer listet das Verzeichnis der externen Symbole auf
FLAG [(I|W|E|S)]	F [(I|W|E|S)]	Der Kompilierer gibt Nachrichten aus, wenn eine Information der genannten Stufe (I, W, E oder S) anfällt
FLOW [(n,m)] | NOFLOW	-	Der Kompilierer erzeugt einen Maschinencode, der die Programmablaufsteuerung für n Eingangsstellen in nicht mehr als m Prozeduren verfolgt; standardmäßig gilt für (n,m) das Wertepaar (25,10)

Parameter (Optionen)	*Abkürzung*	*Bedeutung*
GONUMBER | NOGONUMBER	GN | NGN	Die Objektnachrichten schließen die Zeilennummer der Übersetzung ein
GOSTMT | NOGOSTMT	GS | NGS	Die Objektnachrichten schließen die Statementnummer ein
IMPRECISE | NOIMPRECISE	IMP | NIMP	Aufnahme eines Objektmoduls, der bestimmte Unterbrechungen bei der IBM /370, Modell 195, behandelt
INCLUDE | NOINCLUDE	INC | NINC	Das Statement %INCLUDE ist anwendbar
INSOURCE | NOINSOURCE	IS | NIS	Das Quellenprogramm wird gedruckt, wenn die Option MACRO benutzt wird
LINECOUNT(n)	LC(n)	Anzahl der Zeilen bei den Kompiliererausgaben (standardmäßig 55)
LIST [(n,m)] | NOLIST	-	Der Objektcode wird für die Statements von n bis m gedruckt; bei Angabe von NUMBER erfolgt die Listausgabe für die Zeilen zwischen n und m
LMESSAGE | SMESSAGE	LMSG | SMSG	Langform oder Kurzform der Nachrichten
MACRO | NOMACRO	N | M	Vorübersetzerstatements sind erlaubt
MAP | NOMAP	-	Es wird eine Speicherbelegungsübersicht ausgegeben
MARGINI('c') | NOMARGINI	MI('c') | NMI	Einfügung des Zeichens c vor der linken und nach der rechten Randbegrenzung bei Quellenlisten
MARGINS(m,n [,c])	MAR(m,n [,c])	Randbegrenzungen bei Qellenmoduleingaben (standardmäßig bilt 2,72,1)
MDECK | NOMDECK	MD | NMD	Die Ausgabe des Vorübersetzers wird über SYSPCH ausgegeben
NAME('name')	N('name')	Verwendung des Namens name als Gliedname eines NAME-Statements für den Binder (Linkage Editor)

Parameter (Optionen)	*Abkürzung*	*Bedeutung*
NEST | NONEST	-	Die Listen der Quellenmoduln werden vom Kompilierer durch die Stufennummern für die Blöcke und die DO-Gruppen ergänzt
NUMBER | NONUMBER	NUM | NNUM	Benutzung von Zeilennummern anstelle von Statementnummern bei Nachrichten
OBJECT | NOOBJECT	OBJ | NOBJ	Ausgabe des Objektmoduls auf SYSLIN
OFFSET | NOOFFSET	OF | NOF	Der Kompilierer gibt den Abstand (Offset) jedes Statements von der primären Eingangsstelle aus
OPTIMIZE(TIME|Ø|2) | NOOPTIMIZE	OPT(TIME|Ø|2) | NOPT	OPT(Ø) oder NOPT legt eine schnelle Übersetzung fest; OPT(2) oder OPT(TIME) führt zu einem wirksamen Objektcode
OPTIONS | NOOPTIONS	OP | NOP	Die für den Kompilierer geltenden Optionen werden aufgelistet
SEQUENCE(m,n) | NOSEQUENCE	SEQ(m,n) | NSEQ	Der Kompilierer soll den Quellenmodul auf Sortierfolge in den Spalten m bis einschließlich n prüfen
SIZE(y|yk|MAX)	SZ(y|yk|MAX)	Angabe des für den Kompilierer zur Verfügung stehenden Speicherplatzes
SOURCE | NOSOURCE	S | NS	Auflistung des Quellenmoduls
STMT | NOSTMT	-	Alle Nachrichten schließen die Statementnummer ein
STORAGE | NOSTORAGE	STG | NSTG	Der Kompilierer listet den Speicherbedarf des Objektmoduls
SYNTAX | NOSYNTAX [(W|E|S)]	SYN | NSYN [(W|E|S)]	NSYN bewirkt den Abbruch der Übersetzung nach der Syntaxprüfung, wenn ein Fehler des Grades W, E oder S entdeckt wurde
XREF | NOXREF	X | NX	Der Kompilierer gibt eine Liste aus, in die alle Bezeichnungen aufgenommen sind, zusammen mit den Nummern der Statements, in denen sie auftreten

3.2. FORTRAN-Sprache

Parameter (Option)	*Bedeutung*
LIST | NOLIST	Der Objektmodul wird auf SYSPRINT gelistet
NAME=name	Bei einem Hauptprogramm wird dem Binder (Linkage Editor) ein Name übergeben; standardmäßig wird MAIN als Name genommen
LINECNT=n	Anzahl der Zeilen auf den vom Kompilierer herausgegebenen Listen (standardmäßig 50)
SOURCE | NOSOURCE	Das Quellenprogramm wird aufgelistet
DECK | NODECK	Der Objektmodul wird über SYSPUNCH ausgegeben
LOAD | NOLOAD	Der Objektmodul wird über SYSLIN ausgegeben
MAP | NOMAP	Typ und Speicherplatz jedes im Programm benutzten Namens werden ausgegeben
BCD | EBCDIC	Im Quellenprogramm benutzter Zeichenvorrat
ID | NOID	Interne Statementnummern werden in den Code des Objektprogrammes aufgenommen, wodurch der Programmablauf leichter verfolgt werden kann

3.3. COBOL

Parameter (Option)	*Abkürzung*	*Bedeutung*
SIZE=y|yK	SIZ=y|yK	Speicherplatzmenge, die für die Übersetzung zur Verfügung steht
BUF=y|yK	-	Menge des Speicherplatzes, den der Kompilierer für EA-Puffer benutzen kann
LINECNT=n	CNT=n	Anzahl der Zeilen pro Seite für die Auflistung des Kompilierers (Standard:60)
SOURCE | NOSOURCE	SOU | NOSOU	Auflistung des Quellenmoduls
CLIST | NOCLIST	CLI | NOCLI	Drucken der Auflistung des Objektmoduls in verdichteter Form
DMAP | NODMAP	NMA | NODMA	Ausgabe einer Speicherübersicht des Datenteils
LIB | NOLIB	-	Die Statements BASIS und COPY können benutzt werden
VERB | NOVERB	VER | NOVER	Prozedurnamen und Verbnamen sind in gewisse Listen aufgenommen (CLIST, SUPMAP)
LOAD | NOLOAD	LOA | NOLOA	Der Objektmodul wird über SYSLIN ausgegeben
DECK | NODECK	DEC | NODEC	Der Objektmodul wird über SYSPUNCH ausgegeben
SEQ | NOSEQ	-	Der Kompilierer soll die Reihenfolge der Quellenstatements überprüfen
FLAGW | FLAGE	LAGW | LAGE	FLAGW sorgt für die Auflistung aller Nachrichten; FLAGE listet Warnungen nicht, sondern nur Nachrichten höheren Fehlergrades
SUPMAP | NOSUPMAP	SUP | NOSUP	Unterdrückung der Auflistung des Objektmoduls und der Objektstanzausgabe, wenn Fehler der Stufe E auftreten
SPACE1 | SPACE2 | SPACE3	ACE1 | ACE2 | ACE3	Zeilentransport um 1,2 oder 3 Zeilen bei der Auflistung des Quellenmoduls

Parameter (Option)	*Abkürzung*	*Bedeutung*
TRUNC | NOTRUNC	TRU | NOTRU	Durch TRUNC wird ein Code generiert, der bei Elementen mit COMPUTATIONAL-Angabe linksbündige Ziffern abschneidet, wenn arithmetische Resultate vorliegen oder eine MOVE-Anweisung durchgeführt wird. NOTRUNC unterdrückt diesen Code
QUOTE | APOST	QUO | APO	Anzeige, ob ' oder " benutzt wird
XREF | NOXREF	XRE | NOXRE	Ausgabe einer Zuordnungstabelle für die Daten- und Prozedurnamen
ZWB | NOZWB	-	ZWB bewirkt die Vorzeichenstreichung externer Dezimaldatenelemente bei Vergleichen mit alphamerischen Feldern

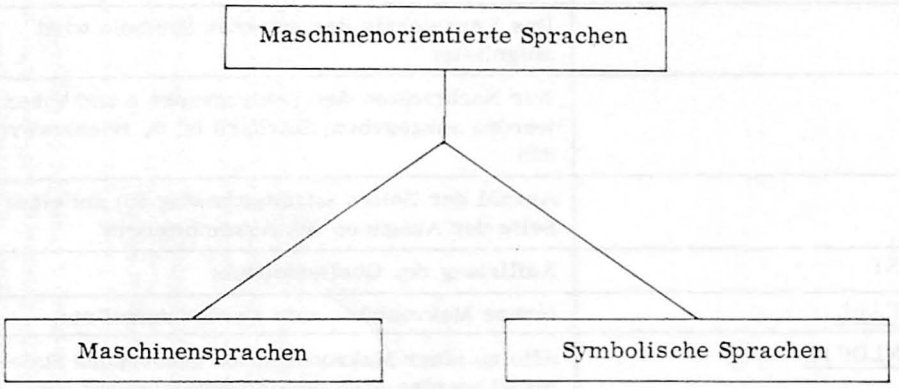
3.4. AL

Parameter (Option)	*Bedeutung*
ALOGIC | NOLAGIC	Bedingte Assembliererstatements werden aufgelistet
ESD | NOESD	Das Verzeichnis der externen Symbole wird aufgelistet
FLAG(n)	Nur Nachrichten des Fehlergrades n und höher werden ausgegeben; Standard ist 0, Höchstwert 255
LINECNT=(n)	Anzahl der Zeilen (standardmäßig 55) auf einer Seite der Ausgaben des Assemblierers
LIST | NOLIST	Auflistung des Quellenmoduls
MCALL | NOMCALL	Innere Makrostatements werden aufgelistet
MLOGIC | NOMLOGIC	Alle zu einer Makrodefinition gehörenden Statements werden nach dem Makrostatement aufgelistet
RLD | NORLD	Auflistung des Verschiebungsverzeichnisses
LIBMAC | NOLIBMAC	Die von Bibliotheken gelesenen Makrodefinitionen werden nach dem END-Statement aufgelistet
XREF(FULL) | XREF(SHORT) | NOXREF	Die Zuordnungstabelle aller benutzten oder definierten Symbole wird ausgegeben; SHORT eliminiert die nicht benutzten Symbole
DECK | NODECK	Der Objektmodul wird über SYSPUNCH ausgegeben
OBJECT | NOOBJECT	Der Objektmodul wird nach SYSGO geschrieben
OBJ | NOOBJ	Abkürzungen von OBJECT bzw. NOOBJECT
TEST | NOTEST	Die Symboltabelle wird als Teil des Objektmoduls gestanzt
ALIGN | NOALIGN	Alle Daten werden auf geeignete Speicherbegrenzungen gelegt. NOALIGN verhindert dies mit Ausnahme von Kanalbefehlswörtern (CCW)
BUFSIZE(MIN) | BUFSIZE(STD)	STD sorgt dafür, daß für den Assemblierer ein Pufferumfang gewählt wird, der zum optimalen Arbeiten führt; MIN wählt den kleinstmöglichen Pufferumfang aus
RENT | NORENT	Nachrichten werden bei möglichen Verletzungen der für das Programmieren geltenden Regeln für die jederzeitige Verwendbarkeit (reentrant code) von Programmoduln ausgegeben
SYSPARM(kette)	Die Kette muß in Hochkommas eingeschlossen werden. Ihr Wert steht dem Assemblierer zur Verfügung, wenn in ihm eine Bezugnahme auf &SYSPARM erfolgt; standardmäßig wird die Nullkette unterstellt

Anhang M: Programmiersprachen

Es gibt zwei Sprachklassen, die selbst wieder nach Unterklassen aufgeteilt werden:

1. Maschinenorientierte Sprachen

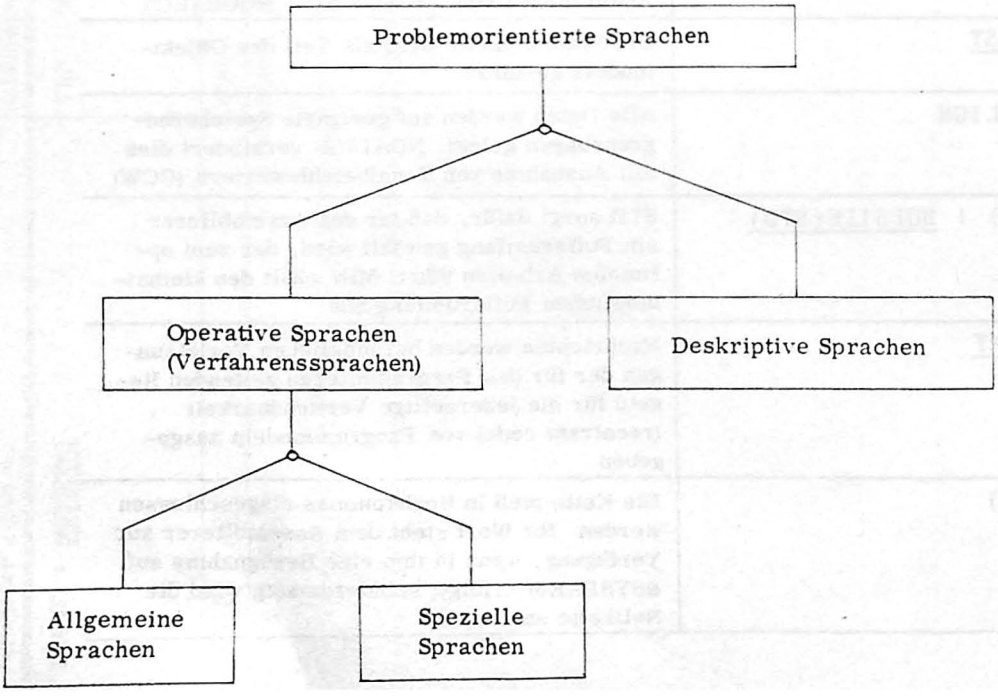


Es ist heutzutage selbstverständlich, daß für jedes Maschinensystem eine symbolische Sprache existiert, mitunter sind auch mehrere, oft noch in verschiedenen Versionen, vorhanden. Die symbolischen Sprachen heißen vielfach *Assemblierersprachen* (engl.: Assembler Languages, abgekürzt AL).

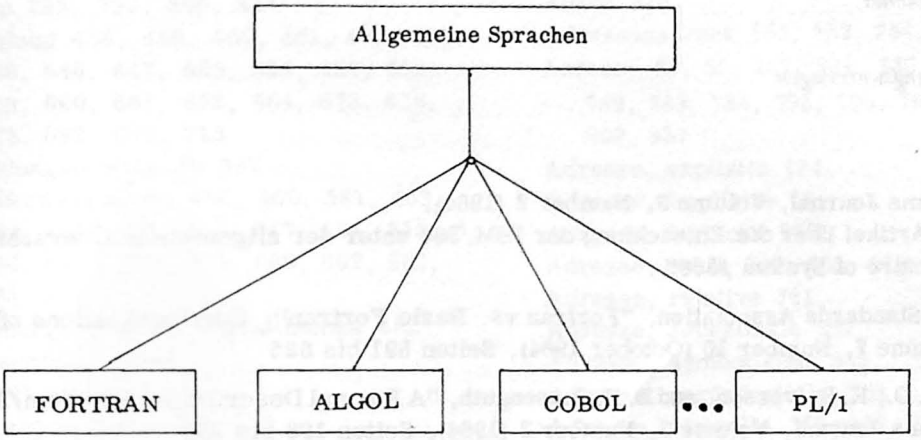
Die meisten dieser Sprachen kommen darüberhinaus noch wesentlichen praktischen Bedürfnissen entgegen, indem sie die Möglichkeit bieten, durch sogenannte „Makrostatements" Instruktions- bzw. Vereinbarungsfolgen aus einer Bibliothek auszuwählen und in ein Programm einzufügen (vor allem für EA-Operationen gebräuchlich). Makrostatements werden oft kurz als Makors bezeichnet.

Die Übersetzungsprogramme heißen *Assemblierer* (englisch: Assembler)

2. Problemorientierte Sprachen

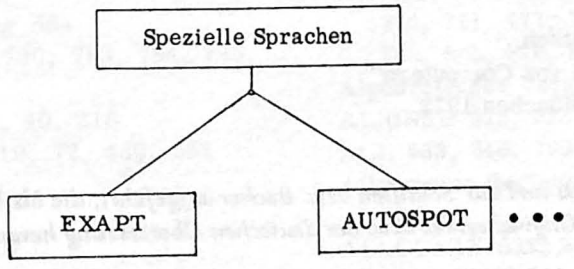


2.1. *Operative Sprachen* dienen zur Beschreibung von Lösungsverfahren. Die Übersetzungsprogramme dieser Sprachen heißen *Kompilierer* (engl.: Compiler).



Es sind hier nur die in der Praxis gebräuchlichen Sprachen aufgeführt

FORTRAN (Formula Translation) ALGOL (Algorithmic Language)	für technisch-wissenschaftliche Anwendungen
COBOL (Common Business Oriented Language)	für kommerzielle (kaufmännische) Anwendungen
PL/1 (Programming Language /One)	für technisch-wissenschaftliche und für kaufmännische Anwendungen



Spezielle Sprachen dienen zur Beschreibung von Lösungsverfahren aus speziellen Anwendungsgebieten, z.B. EXAPT und AUTOSPOT zur numerischen Steuerung von Werkzeugmaschinen.

2.2. *Deskriptive Sprachen* dienen zur Beschreibung einer bestimmten Aufgabenstellung. Die Übersetzungsprogramme für diese Sprachen werden allgemein als *Generatoren* bezeichnet.

In der Praxis hat sich vor allem die RPG-Sprache (Report Program Generator) bewährt, die es gestattet, Informationen aufzunehmen, zu verarbeiten und auszugeben.

Literaturverzeichnis

1. IBM - Handbücher

2. IBM - Schulungsunterlagen

3. Artikel

a) IBM Systems Journal, Volume 3, Number 2 (1964).
Enthält 5 Artikel über die Entwicklung der IBM/360 unter der allgemeinen Überschrift: "The Structure of System /360"

b) American Standards Association, "Fortran vs. Basic Fortran", Communications of the ACM, Volume 7, Number 10 (October 1964), Seiten 591 bis 625

c) Falkoff, A. D., K. E. Iverson, and E. H. Sussenguth, "A Formal Description of System/360", IBM Systems Journal, Volume 3, Number 3 (1964), Seiten 198 bis 263

4. Bücher

a) Farina, Mario V.
"FORTRAN IV: Self Taught"
Prentice-Hall, Inc., 1966, 448 Seiten

b) Mc Cracken, Daniel D.
"A Guide to FORTRAN IV Programming".
John Wiley and Sons, Inc., 1965, 151 Seiten

c) Organick, Elliott I.
"A FORTRAN IV Primer"
Addison-Wesley Publishing Company, Inc., 1966, 263 Seiten

d) Pollack, Seymour V.
"A Guide to FORTRAN IV"
Columbia University Press, 1965, 260 Seiten

e) Rüdiger und Anna Gritsch
"Das Programmieren von Computern"
Carl Hanser Verlag München 1972

In diesem Literaturverzeichnis sind nur Schriften bzw. Bücher aufgeführt, die als Unterlage bei der Erstellung des amerikanischen Originalwerkes bzw. der deutschen Übersetzung herangezogen wurden.

A Programming Language 325
A-Typ 795, 796, 806, 814
Abbildung 458, 459, 460, 461, 462, 463, 480, 646, 647, 655, 656, 657, 658, 659, 660, 661, 662, 664, 675, 676, 678, 682, 691, 713
Abbildungsbibliothek 347
Abbildungszeichen 459, 460, 461, 462, 480, 481, 523, 646, 647, 654, 655, 656, 657, 658, 659, 660, 662, 663, 664, 713
Abbildungszeichen, statistisches 462
Abbrechfehler 529
ABEND 934
Abfertigungspriorität 355
Abgang 373, 374
Abkürzung 430, 431
Ablagefach 333, 406, 468, 691, 827, 832, 895
Ablagefachauswahl 691
Ablaufvariable 451, 452, 453
Abrechnungsinformation 352, 357
Abrechnungsnummer 352, 353
Abrechnungsoperand 352, 353
Abschließen 558, 955
Abschließen von Dateien 423, 424, 470, 650, 699, 841, 900
Abschließen, explizites 423
Abschließen, implizites 423, 470, 473
Abschnitt, gemeinsamer 328, 810, 811
Abschnittsname 329
Absolute Sprache 768
Absolute Verzweigung 554
Absoluter Ausdruck 780, 783, 784, 795, 796, 799
Absoluter Betrag 18, 40, 216
Absoluter Wert 18, 19, 72, 432, 562
Abstand 527
Abtasten 368
Abtastmodus 893, 907
Abwärtsverträglichkeit 264
ACB 950, 952, 955, 956, 962, 964
ACCEPT 695, 696
ACCES 692, 693
ADD 650, 669, 670, 671
ADDBUFF 466, 489
Addition 399, 442, 444, 529, 549, 550, 660, 669, 670, 672, 714, 780
Addition von Matritzen 33
Additionsinstruktionen 64, 85, 219, 220, 221, 224, 236
Additives Dateiattribut 465
ADDR 476, 477
Adkon 328
Adressausdruck 885, 852, 954, 958
Adresse 52, 55, 267, 328, 332, 779, 781, 782, 783, 784, 795, 796, 797, 808, 809, 827
Adresse, explizite 784
Adresse, implizite 784
Adresse, logische 267
Adresse, reale 267, 269, 279
Adresse, relative 781
Adresse, sedezimale 797
Adresse, symbolische 852
Adresse, veränderliche 768
Adresse, virtuelle 267, 268, 270, 273
Adreßkomponente 371
Adreßkonstante 328, 329, 795, 796, 797, 806, 808, 814, 884
Adreßliste 944
Adreßmarke 197, 198
Adreßmodifikation 253
Adreßrechnung 253
Adreßteil 797
Adreßumsetzspeicher 272
Adreßumsetzung, dynamische 264, 265, 266, 270, 272, 273
Adreßzähler 781, 782, 807, 808, 809, 810
ADVANCING 697
Äußere DO-Anweisung 585
AFTER 697
Aktivieren 505, 509
AL 324, 326, 336, 367, 376, 596, 769, 770, 771, 777, 781, 783, 784, 785, 786, 805, 816, 971, 979
Algebraischer Vergleich 409
ALIGNED 522, 523, 524
ALL 639, 666, 709
Allgemeine Register 55, 60, 75, 80, 781, 826
ALLOCATE 533, 534, 535, 536
ALLOCATION 536
ALOG 551
ALOGIO 551
Alphaarbeitsweise 346
Alphabet 388, 547, 583
ALPHABETIC 713
Alphabetische Daten 654, 658
Alphabetische Klasse 713
Alphabetische Zeichen 246
Alphabetisches Datenelement 654, 656, 664, 665, 666, 682, 695, 712, 713
Alphabetisches Zeichen 388, 390, 394, 547, 637

Alphamerische Daten 655
Alphamerisches Zeichen 388
Alphanumerische Daten 570, 572, 655, 656
Alphanumerische Formatangabe 570, 571
Alphanumerische Zeichen 246
Alphanumerisches Datenelement 655, 656, 658, 664, 665, 666, 682, 687, 695, 712, 713
Alphanumerisches Zeichen 388, 389, 570, 637
ALTER 707, 708
ALTERNATE 692
Altes PSW 90, 91, 247, 248, 270, 280, 885
AMASTCAT 384
American National Standards 632
AMS 963, 964
And 25, 26, 707, 715, 716
Anfangsetikette 312
Anfangspunkt 854
Anfangswert 431, 432, 438, 439, 440, 441, 501, 504, 505, 521, 584, 586, 596, 647, 676, 678, 683, 807, 992
Anforderungsparameterliste 952, 953, 956, 958, 960, 961, 962
Anführungszeichen 341, 353, 357, 358, 388, 391, 406, 461, 571, 637, 638, 639, 779, 789
Angabe 405, 406, 409, 415, 423, 522
Anormaler Ausgang 509, 513, 971, 972
ANS 325, 405, 422, 632, 637
ANS-COBOL 632
Ansprung 781
Anweisung 402, 403, 404, 405, 408, 409, 411, 412, 414, 415, 421, 422, 423, 424, 426, 440, 450, 451, 452, 466, 469, 470, 473, 477, 478, 479, 481, 482, 483, 485, 486, 488, 489, 490, 499, 507, 509, 511, 512, 514, 515, 522, 529, 532, 533, 534, 535, 536, 553, 554, 558, 562, 570, 583, 584, 585, 588, 589, 591, 595, 634, 649, 674, 715
Anweisung, bedingte 671
Anwendungsprogramm 326, 327, 375
Anzeige 695
Anzeigeleiste im Kanal 108, 121, 122, 125
APL 325
Apostroph 388
APPLY 693, 694
Arbeitsdatei 331, 344, 348, 367, 485, 838, 839, 840, 849, 850, 851, 852, 853, 854, 855, 872, 873, 978, 979, 983, 984, 989
Arbeitsdatenträger 360
Arbeitsende 75
Arbeitsgebiet 84, 306, 328, 349, 367, 537, 634, 635, 771, 772, 776, 825, 826, 865, 866, 868, 897, 904, 908, 912, 953, 960, 962
Arbeitsgebietlänge 908
Arbeitsgruppe 970
Arbeitsmodus 832
Arbeitsspeicher 52, 351, 375, 677, 681, 683, 702, 703, 934
Arbeitsspeicherkapitel 644, 645, 647, 649, 651, 680
Arbeitsvorbereitung 344
ARE 644, 692
AREA 523, 534, 692, 694
AREAS 692
Argument 432, 433, 434, 476, 481, 505, 507, 508, 510, 511, 512, 516, 537, 538, 551, 818, 819, 944, 974, 975, 986
Argument einer komplexen Zahl 41
Argument, komplexes 432
Argumentenliste 433, 505, 818, 819, 842, 946
Arithmetische Daten 392, 395, 399, 409, 410, 414, 418, 458
Arithmetische Ergibtanweisung 443, 552, 572, 583
Arithmetische Instruktion 545, 547, 552
Arithmetische Konstante 529
Arithmetische Operation 43, 57, 401, 402, 411, 441, 442, 443, 444, 529, 550, 650, 658, 660
Arithmetische Operatoren 31, 640
Arithmetische Variable 398, 443, 499
Arithmetischer Ausdruck 399, 400, 401, 443, 462, 549, 651, 671, 711, 716
Arithmetischer Typ 441, 451, 482
Arithmetisches Attribut 397
Arithmetisches Datenelement 410
Arithmetisches Statement 669, 670, 671, 678
ASA 405, 855
ASA-Steuerzeichen 405, 422, 698, 833, 855
ASCII 311, 312, 317, 320
Assembler 323, 364, 789, 875
Assembler Language 324, 367, 376, 769, 777
Assembler/VS 325
Assemblersprache 510, 769
Assemblierer 323, 325, 326, 769, 770, 774, 775, 776, 778, 779, 780, 781,

782, 783, 784, 785, 786, 790, 791, 793, 795, 796, 797, 798, 799, 801, 802, 803, 804, 805, 807, 808, 809, 810, 811, 873, 892, 966, 976
Assemblierer, Steueranweisung für den 777, 778, 801, 802, 803, 804, 805, 806, 807
Assembliereranweisung 776, 802, 807
Assemblierersprache 324, 339, 768, 769, 774, 777, 963
ASSEMBLY 336
ASSGN 331, 332, 333, 334, 335, 338, 339, 340, 345, 652
ASSIGN 589, 692
ASSOCIATE 467
Assoziationsspeicher 272
Assoziierte Datei 467
Asynchroner Zugriff 954
Asynchronische Operation 953
AT 649, 698, 699
ATRB 957
Attribut 362, 394, 395, 396, 397, 398, 400, 401, 405, 409, 414, 420, 421, 423, 430, 431, 432, 433, 440, 441, 442, 443, 451, 459, 461, 464, 465, 468, 469, 470, 476, 477, 478, 480, 481, 484, 485, 486, 488, 489, 490, 500, 504, 505, 506, 507, 508, 510, 511, 512, 519, 520, 521, 522, 523, 524, 529, 530, 531, 534, 536, 957
Attribut, arithmetisches 397
Attribute, Zusammenfassung 432
Attributmenge 396, 397
Attributzuordnung 396
Aufbereitete alphanumerische Daten 656
Aufbereitete numerische Daten 661, 662, 663, 664
Aufbereitetes alphanumerisches Datenelement 656, 657, 664, 666, 682, 712
Aufbereitetes Feld 462
Aufbereitetes numerisches Datenelement 661, 662, 663, 664, 665, 666, 682, 712
Aufbereitung 253, 254, 255, 461, 462, 656, 662, 884
Aufbereitung von Daten 421
Aufbereitungszeichen 254, 255
Aufdatieren 484, 849, 893, 903, 907, 954, 955, 962, 967, 968
Aufdatierung 864, 872, 909, 924, 925, 931, 963, 966
Auffüllung 792, 793, 795
Aufgerufene Programmeinheit 299, 817, 818, 819, 944
Aufgezeichneter Schlüssel 487
Auflistung 803, 811
Aufruf 299, 505, 506, 508, 509, 511, 512, 514, 531, 974, 975, 986
Aufrufende Programmeinheit 299, 816, 817, 818, 819, 943, 946
Aufrufende Prozedur 507, 508, 510, 512
Aufrufendes Programm 507, 510, 513
Aufrufinstruktion 299
Aufrufkette 816
Aufsuchen 201
Auftrag 327, 333, 334, 336, 348
Auftragsende 826
Auftragsname 336
Auftragssteuerung 327, 335, 336
Auftragsteil 336
Aufwärtsverträglichkeit 264
Aufzeichnungsdichte 333, 335, 350, 895
Aufzeichnungsmodus 990
Aufzeichnungssatz 308
Aufzeichnungsschlüssel 487
Ausbaustufe 325
Ausdruck 399, 400, 401, 409, 414, 415, 419, 421, 423, 432, 434, 443, 444, 450, 451, 454, 457, 469, 470, 481, 482, 483, 485, 486, 488, 489, 490, 505, 507, 508, 512, 520, 528, 531, 533, 549, 550, 551, 554, 639, 669, 671, 684, 778, 780, 784, 786, 787, 795, 796
Ausdruck, absoluter 780, 783, 784, 795, 796, 799
Ausdruck, arithmetischer 399, 400, 401, 443, 462, 549, 651, 671, 711, 716
Ausdruck, einfacher 780, 781
Ausdruck, logischer 716
Ausdruck, verschieblicher 780, 781, 795, 796, 808
Ausdruck, zusammengesetzter 780
Ausführbares Programm 330, 336
Ausführen 257, 258, 283, 363, 364
Ausführungsform von Makros 930
Ausführungsstep 365
Ausführungszeichen 675
Ausführungszeit 770
Ausgabe 43, 74, 98, 345, 404, 405, 409, 414, 415, 416, 417, 418, 419, 420, 423, 441, 447, 463, 474, 478, 503, 514, 515, 557, 559, 570, 574, 577, 579, 599, 631, 694, 698, 770, 823, 832, 836, 844, 845, 859, 913
Ausgabe, formatgesteuerte 422
Ausgabeanweisung 562
Ausgabeanweisung, formatgesteuerte 415
Ausgabedatei 405, 421, 464, 466, 649, 692,

828, 830, 840, 850, 859, 872, 978, 983, 984, 989
Ausgabedaten 341, 349
Ausgabedatenliste 415
Ausgabedatenmenge 359, 360, 987
Ausgabedatenträger 413
Ausgabefehler 910
Ausgabefeld 416, 418
Ausgabegebiet 826, 832, 840, 842, 844, 851
Ausgabeklasse 350, 359
Ausgabeliste 802
Ausgabeoperation 367, 837
Ausgabeprozedur 991
Ausgabepuffer 478, 694, 846, 906
Ausgabesatz 474, 571, 572, 573, 833
Ausgabeschreiber 349, 350
Ausgabeschreiberklasse 354
Ausgabestatement 650
Ausgabezeile 359, 405, 576
Ausgang 960, 971, 972, 979, 984, 987
Ausgang, anormaler 509, 513, 971, 972
Ausgangsdatei 405
Ausgangsleitung 137
Ausgangsliste 895
Ausgangsroutine 941, 951, 952, 961, 963
Ausgerichtete Bitkette 392
Auslauf 827
Ausnahmebedingung 881, 925, 928
Ausrichtung 522, 523
Ausrüstung 633
Ausrüstungsfehler 880
Ausrüstungskapitel 690, 691
Austauschgebiet 345, 346
Auswahlleitung 137
Auswahlsatz 642, 692
Ausweichspur 196
Autolink 329
AUTOMATIC 500, 501, 503, 510, 535
Automatische Programmierung 769
Automatische Speicherklasse 505, 531, 535, 973
Automatisches Binden 329

B-Typ 792, 793, 798, 799
BACKSPACE 593, 600
Band 367, 700, 823, 827, 852, 872, 984
Bandbegrenzungszeichen 314
Bandetikette 312-317
Bandkennsatz 312-317, 341
Bandmarke 183, 314, 315, 335, 362, 406, 468, 592, 694, 827, 828, 840, 841, 852, 854, 906
Bandnachsatz 878
Bandrolle 333
BASED 476, 500, 521, 532, 534, 535,
BASIC FORTRAN-Sprache 582
Basis 392, 393, 395, 399, 400, 401, 432, 476, 477, 546, 672
Basisadresse 60, 74, 769, 796
Basisbezogene Speicherklasse 535
Basisbezogene Struktur 477, 532, 534, 535
Basisbezogene Strukturvariable 533
Basisbezogene Variable 476, 477, 478, 527, 531, 533, 534
Basisname 520
Basisregister 60, 74, 230, 280, 282, 297, 771, 772, 781, 782, 783, 784, 808, 813, 814, 913
Basissteuerung 92, 110, 230, 239, 243, 244, 248
Basisvariable 476, 520, 521
Basiszeichen 648
Basiszugriff 367
Basiszugriffsmethode 893, 917, 918, 919
BCD 44, 45
BDAM 893, 896, 918, 919, 923, 924, 926, 927, 928, 929, 931
Bedienungspersonal 326, 327, 768, 769, 937
Bedingte Anweisung 671
Bedingte Verzweigung 70, 73
Bedingung 355, 357, 426, 427, 428, 429, 430, 434, 459, 473, 481, 482, 483, 495, 499, 500, 507, 513, 514, 515, 528, 535, 710, 711, 714, 715, 716
Bedingung, verbundene 715, 716
Bedingungsname 684, 714, 715
Bedingungsnamenprüfung 714
Bedingungspräfix 429
Bedingungsschlüssel 70, 71, 72, 73, 85, 86, 87, 89, 94, 109, 122, 124, 126, 127, 128, 132, 133, 134, 217, 219, 222, 226, 230, 236, 250, 259, 273, 276, 277, 355, 357, 917, 942, 957, 958
Befehl 98, 99, 367, 406, 823, 826
Befehlskettung 105, 117, 201, 917
Befehlsschlüssel 103, 104, 105, 116, 131, 799
Befehlssteuerblock 876, 878, 881, 882
Befehlswiederholung 122
BEFORE 697
BEGIN 498, 500
BEGIN-Block 498, 499, 500, 509
Begrenzung 524
Begrenzungszuordnung 798, 799, 807, 899
Behälter 368
Beigeordnete Variable 594, 595

Bemerkung 332, 352, 675, 774
Benutzerausgang 979
Benutzerbedingung 428
Benutzerkatalog 384
Benutzerkennsatz 313, 362, 839, 849, 850, 858
Benutzerpartition 975
Benutzerprogramm 325, 339, 340, 345, 353, 365, 367, 374, 963, 979
Benutzerroutine 841, 842, 863, 987
Benutzervorsatz 878
Bereich 338, 342, 343, 344, 361, 369, 373, 379, 382, 435, 436, 437, 438, 439, 440, 450, 453, 456, 468, 508, 514, 519, 520, 521, 522, 525, 533, 581, 584, 598, 599, 678, 679, 680, 681, 682, 687
Bereich für Informationsaustausch 883,887
Bereich, eindimensional 679
Bereich, zweidimensional 679
Bereichselement 435, 436, 437, 438, 450, 453, 456, 457, 476, 520, 588, 589, 599, 679, 681, 687
Bereichserklärung 435
Bereichsname 435, 436, 438
Bereichsnummer 343
Bereichsumfang 453
Bereichsvariable 435, 437, 455, 509, 581, 582
Beschriftungszeile 14
Betriebssystem 323, 325, 326, 327, 329, 330, 331, 334, 337, 341, 345, 347, 348, 351, 352, 353, 354, 358, 360, 363, 367, 368, 369, 379, 380, 382, 383, 384, 386, 393, 396, 398, 402, 403, 404, 405, 406, 407, 409, 423, 424, 465, 466, 467, 468, 469, 470, 471, 486, 487, 488, 489, 505, 514, 515, 516, 522, 537, 557, 570, 595, 632, 641, 642, 650, 651, 676, 691, 692, 694, 695, 696, 699, 706, 715, 774, 777, 793, 795, 798, 802, 803, 805, 806, 808, 811, 816, 819, 823, 841, 882, 892, 895, 917, 949, 950, 951, 952, 954, 955, 958, 963, 964, 979, 990
Bezeichnung 390, 398, 399, 428, 430, 434, 456, 498, 500, 511, 641, 666, 669, 671, 715
Bezeichnung, externe 389, 390, 428
Bezeichnung, interne 389, 511
Bibliothek 304, 326, 327, 329, 330, 340, 341, 347, 348, 357, 359, 363, 365, 536, 537, 551, 805, 869, 884, 892, 946
Bibliothek ausführbarer Programme 330, 336, 340
Bibliothek der Lademoduln 336, 340
Bibliothek für COBOL 347
Bibliothek für FORTRAN 347
Bibliothek für Objektmoduln 869
Bibliothek für PL/1 347
Bibliothek für verschiebliche Programme 869
Bibliothek für virtuelle EA 347
Bibliothek, private 330
Bibliotheksdatei 537
Bibliotheksdatenmenge 814
Bibliotheksglied 537
Bibliotheksname 537
Bibliotheksprogramm 386
Bibliotheksverzeichnis 304
Bildschirm 836
Billings, Dr. John 1
BIN 368, 431
Binäranzeiger 44, 49, 50
Binäres Lesen 833
Binärmuster 182
Binärzahl 21
Binärzeichen 980
BINARY 395, 396, 397, 398, 401, 409, 427, 431, 433, 437, 443, 506, 507, 523, 529, 530, 531, 532
Binden 337, 363, 364
Binden, automatisches 329
Binder 327, 328, 329, 330, 336, 338, 347, 364, 365, 501, 803, 805, 808, 810, 813, 814, 869
Bindername 336
Binderstep 365
Bindestrich 353, 357, 380, 635, 637, 641, 675
BISAM 893, 896, 918, 919, 921, 924
Bit 44, 46, 392, 395, 420, 444, 458, 522, 523, 529, 530, 715, 779, 791
Bitkette 391, 392, 395, 401, 406, 409, 411, 417, 441, 442, 443, 444, 481, 483, 521, 524, 529
Bitkette, ausgerichtete 392
Bitkette, nicht ausgerichtete 392
Bitkettendaten 409
Bitkettenvariable 440, 442
Bitklassendaten 521
Bitlängenspezifikation 792, 793
Bitmuster 47, 182, 227, 433, 482
Bitstelle 44, 45, 46, 48, 51, 55, 88, 100, 108, 109, 110, 113, 116, 121, 216, 234, 241, 267, 345, 371, 373, 392, 395, 410, 427, 442, 443, 482, 523, 639, 793, 794, 795, 799
Blank 388

BLANK 682
BLKSIZE 405, 465
Block 183, 294, 305, 308, 320, 360, 367, 369, 374, 376, 429, 498, 499, 500, 501, 503, 504, 515, 522, 644, 694, 823, 828, 829, 842, 849, 854, 866, 882, 905, 910, 917, 922, 923, 925
Block, interner 500
Blockadresse 926
Blocken 823
Blockende 500
Blockkontrollfeld 308
Blocklänge 305, 311, 360, 405, 406, 422, 466, 897, 908
Blocklänge, physikalische 644
Blockmultiplexkanal 99, 100, 102, 265
Blocknummer 369
Blockumfang 305
Blockung 183, 305, 369
Blockungsfaktor 305, 311, 841, 865, 882
Blockungsfehler 909
Blockzähler 854, 917
Bogenmaß 433
Bogenmaß von Winkeln 39
Boolesche Arithmetik 25
Boolesche Operatoren 25, 26
Boolesche Variablen 25
BPI 333, 350
Bruch 30
BS 122, 226, 236
BSAM 893, 896, 918, 919, 921, 924, 928
Buch 805
Buchname 805
Buchstabe 2, 46, 336, 340, 352, 388, 389, 393, 397, 459, 460, 482, 506, 507, 547, 548, 637, 638, 641, 643, 656, 713, 778, 779
BUFFERED 464, 478, 484, 485, 490
BUFFERS 405, 465, 485
BUFFERSPACE 381, 382
BUILD 900, 901, 919, 924, 928
BUILDREC 901
BUILTIN 433
Burst Mode 99
BY 451, 452, 457, 458, 650, 670
BY NAME 457, 458
Byte 44, 45, 46, 52, 53, 59, 79, 81, 83, 86, 88, 89, 100, 104, 108, 112, 117, 226, 241, 267, 345, 349, 368, 369, 370, 371, 373, 374, 376, 377, 378, 390, 392, 395, 458, 466, 467, 482, 487, 488, 489, 521, 522, 523, 525, 526, 527, 571, 594, 636, 639, 646, 654, 656, 658, 659, 663, 677, 683, 684, 695, 697, 777, 779, 781, 786, 789, 790, 791, 792, 793, 796, 798, 799, 807, 827, 842, 854, 866
Byteadresse, relative 377, 958, 959
Bytegrenze 523, 524, 526, 791, 792, 793, 794, 795
Bytemultiplexkanal 99, 100, 124, 126

C-Typ 789, 791, 792, 798, 799, 810
CALL 511, 512, 513, 515, 590, 591, 592, 888, 944, 945, 946
CANCEL 884, 886
CAT 329
CATALOG 384
CAW 107, 109, 114, 119, 123, 130, 131, 132, 824, 894
CC-Anzeiger 105, 106, 117
CCB 876, 878, 881
CCW 104, 105, 106, 107, 109, 110, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 130, 201, 271, 312, 799, 800, 876, 877, 881, 894, 917, 931
CD-Anzeiger 105, 106
CDMOD 869, 870, 871, 876
CHANGED 706
Channel Command Word 104, 105, 106, 107, 109, 110, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 130, 201, 271, 312, 799, 800, 876, 877, 881, 894, 917, 931
Channel Logout 108
CHAR 395, 431, 458, 519, 521, 530
CHARACTER 383, 395, 420, 430, 431, 458, 523, 530, 531, 532
CHARACTERS 644, 690
Charakteristik 25, 216, 219, 222, 244
CHECK 429, 852, 853, 854, 893, 897, 916, 918, 922, 924, 925, 926, 928, 950, 958
Chefprogrammierer 970, 971, 973
CIDF 376
CLB 329
CLOSE 335, 423, 424, 470, 473, 478, 486, 489, 534, 636, 650, 699, 700, 823, 825, 826, 829, 834, 841, 852, 857, 877, 892, 903, 904, 908, 910, 919, 924, 928, 930, 935, 950, 955
CLUSTER 381, 382, 383
Cluster 378, 381, 382, 383
CNOP 809, 810
CNTRL 827, 835, 849, 855, 858, 893, 920
COBOL 324, 325, 326, 336, 339, 347,

364, 367, 369, 376, 466, 490, 522, 596, 631, 632, 635, 637, 639, 641, 645, 650, 651, 654, 656, 660, 665, 685, 709, 716, 770
COBOL-Programm 632, 634, 635, 639, 641, 642, 651, 660, 674, 675, 681, 690, 698, 701, 709, 713, 988, 989, 990, 991, 992
COBOL-Statement 635, 640
COBOL-Wort 637, 641, 652
COCR 370
Codieren 970
Codierung 426, 429, 430, 431, 463, 529, 550, 812, 819, 848, 884
COL 431
COLBIN 468
Columbia 1
COLUMN 419, 431
COM 811, 812
COM-REG 676
COMMA 690
Command Control Block 876, 878
Common Section 810
COMP 646, 683
Compiler 324, 325
COMPLEX 395, 396, 397, 398, 401, 409, 431, 459, 482, 590
COMPUTATIONAL 635, 646, 648, 659, 660, 663, 665, 671, 676, 680, 683
COMPUTE 669, 670, 671, 672, 686
Computer 9, 43, 44, 45, 46, 55, 56, 90, 91, 92, 98, 100, 114, 120, 124, 222, 236, 237, 248, 252, 254, 259, 264, 267, 274, 285, 290, 300, 323, 324, 325, 336, 348, 374, 375, 386, 387, 522, 543, 563, 570, 591, 631, 632, 634, 652, 769, 825, 876
Computergrundsprache 768
Computing Tabulating Recording Co. 1
Comreg 883
CONFIGURATION 690
CONSECUTIVE 465
CONSOLE 691, 695, 696
CONTAINS 644, 693, 694
CONTINUE 584
Control Interval Definition Field 376
Control Section 328, 501, 810
CONTROL INTERVALSIZE 381, 382
CONTROLLED 536
CONV 431
CONVERSION 427, 428, 431, 434, 459, 482, 483, 495
COPY 408, 413, 414, 415, 481, 805
Cosecans 39
Cosinus 39
Cotangens 39
CPL X 431
CSECT 328, 848
CSW 107, 109, 110, 112, 113, 117, 120, 121, 124, 125, 128, 129, 131, 238, 877, 917, 918
CURRENCY 690, 691
CURRENT-DATE 675, 676
Cylinder 368
Cylinder Overflow Control Record 370
CYLINDERS 379, 380, 381, 382

D-Typ 795, 798, 799
DA 342, 643
DAMOD 869, 874
Darstellungsform 678
DATA 381, 413, 422, 427, 644, 645
Data Definition 358
DATA DIVISION 633, 644, 645
Data Event Control Block 917
Data Mode 902
DATE 696
DATE-COMPILED 642
Datei 303, 342, 344, 365, 367, 403, 405, 407, 408, 413, 415, 421, 423, 424, 427, 464, 465, 466, 468, 469, 470, 471, 473, 478, 484, 485, 486, 487, 488, 489, 490, 491, 494, 495, 500, 514, 515, 522, 537, 557, 558, 580, 592, 593, 594, 596, 600, 631, 634, 636, 642, 644, 645, 647, 649, 650, 651, 676, 686, 692, 693, 694, 695, 697, 698, 699, 700, 702, 824, 830, 831, 833, 835, 837, 838, 839, 841, 841, 842, 848, 850, 852, 853, 854, 855, 858, 859, 861, 863, 864, 866, 867, 869, 872, 874, 878, 893, 897, 900, 901, 903, 908, 909, 919, 920, 922, 923, 926, 930, 954, 955, 962, 989, 990
Datei, abschließen 423, 424, 470, 650, 699, 841, 900
Datei, assoziierte 467
Datei, direkt organisiert 857
Datei, eröffnen 423, 424, 469, 470, 649, 650, 698, 699, 825, 841
Datei, gestreut organisiert 857
Datei, indexsequentiell organisiert 863, 882
Datei, kombinierte 830, 831, 832, 833
Datei, konsekutiv organisiert 592, 595, 643, 698, 857
Datei, logisch fortlaufend organisiert 863, 882, 908

Datei, regional organisiert 857
Datei, starr fortlaufend organisiert 595, 596, 857, 919
Datei, ungepufferte 490
Datei, verknüpfte 833, 835
Dateianfang 868
Dateiattribut 464, 466, 484, 488, 984
Dateiattribut, additives 465
Dateibereich 864
Dateibeschreibung 406, 644, 645
Dateidefinition 824, 825, 830, 834, 836, 838, 839, 842, 852, 855, 857, 859, 860, 863, 868, 869, 875, 877, 878
Dateien, erstellen von 963
Dateiende 199, 359, 369, 403, 405, 592, 698, 699, 825, 831, 836, 850, 853, 854, 855, 867, 879, 882, 902, 904, 929, 951, 952, 959, 960
Dateiendebedingung 332, 338, 403, 408, 825, 853, 882, 952
Dateiendesatz 369, 861, 862, 866
Dateienkapitel 699
Dateierklärung 488, 490
Dateieröffnung 423, 424, 469, 470, 649, 650, 698, 699, 825, 841
Dateikapitel 644, 649, 650, 651, 676, 677
Dateimaske 207, 208, 210
Dateiname 341, 358, 379, 405, 406, 407, 408, 413, 415, 423, 426, 427, 464, 465, 469, 470, 478, 484, 485, 486, 488, 490, 500, 537, 557, 649, 650, 691, 693, 698, 699, 852, 853, 857, 868, 869, 878, 984
Dateinummer 593
Dateiorganisationsform 490, 491
Dateischutz 184, 211
Dateisteuerungsparagraph 692, 693
Dateiverwendung 840, 851
Dateiverwendungsattribut 488, 490
Daten 56, 59, 62, 206, 339, 340, 342, 344, 349, 371, 381, 382, 386, 390, 399, 407, 408, 413, 414, 415, 421, 427, 441, 458, 461, 471, 475, 478, 480, 481, 495, 522, 545, 547, 552, 557, 593, 598, 631, 677, 678, 694, 853, 860, 861, 866, 926
Daten, alphabetische 654, 658
Daten, alphamerische 655
Daten, alphanumerische 570, 572, 655, 656
Daten, arithmetische 392, 395, 399, 409, 410, 414, 418, 458
Daten, aufbereitete alphanumerische 656
Daten, aufbereitete numerische 661, 662, 663, 664
Daten, Aufbereitung 421
Daten, dezimale 392
Daten, duale 392
Daten, komplexe 392, 418, 458
Daten, numerische 427, 481
Daten, reelle 392
Datenanordnung 463
Datenattribut 512
Datenaustausch 818, 819
Datenbereich 343
Datenbeschreibung 645, 647, 677, 682
Datenbestand, indexsequentiell organisiert 343
Datenbyte 377
Datendefinition 358, 848, 849
Datenelement 328, 413, 415, 416, 452, 453, 454, 463, 481, 486, 522, 523, 529, 644, 645, 646, 647, 649, 650, 654, 655, 656, 658, 659, 660, 661, 662, 665, 666, 669, 671, 677, 679, 681, 682, 683, 685, 686, 687, 699, 706, 707, 711, 713, 714
Datenelement, alphabetisches 654, 656, 664, 665, 666, 682, 695, 712, 713
Datenelement, alphamerisches 655, 656, 658, 664, 665, 666, 682, 687, 695, 712
Datenelement, alphanumerisches 713
Datenelement, arithmetisches 410
Datenelement, aufbereitetes alphanumerisches 656, 657, 664, 666, 682, 712
Datenelement, aufbereitetes numerisches 661, 662, 663, 664, 665, 666, 682, 712
Datenelement, numerisches 646, 661, 662, 664, 687
Datenende 338
Datenereignissteuerblock 917, 918, 919, 920, 921, 925, 928, 929
Datenerfassung 543
Datenerklärung 674
Datenfehler 137, 243, 862, 880
Datenfeld 199, 200, 206, 318, 427, 474, 483, 861, 862
Datenfeld, numerisches 460, 461
Datenflußplan 290-293
Datenformat 590
Datenformatelement 415, 419
Datengebiet 379, 699
Datengeneration 536
Datengesteuert 463, 481, 483
Datengesteuerte EA 413, 414

Datengesteuerte Eingabe/Ausgabe 413, 414, 415
Datengruppe 677
Datengruppierung 522
Datenkarte 339, 476
Datenkettung 105, 116, 117, 119, 917
Datenkomponente 382, 948, 949
Datenliste 408, 413, 415, 419, 421, 422, 452, 453, 454, 480, 481, 558, 559, 571, 572, 574, 582, 583, 596, 597, 598, 599
Datenmanagement 327
Datenmenge 303, 304, 314, 319, 327, 338, 341, 342, 343, 344, 347, 348, 354, 358, 359, 360, 361, 362, 363, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 423, 466, 467, 468, 470, 471, 480, 484, 485, 486, 487, 488, 489, 514, 558, 595, 631, 642, 692, 694, 823, 892, 894, 897, 903, 904, 923, 924, 931, 948, 949, 950, 960
Datenmenge in Schlüsselfolge 375
Datenmenge in Zugangsfolge 375
Datenmenge, gestreut organisiert 892, 894
Datenmenge, indexsequentiell organisiert 361, 376, 892, 894, 897
Datenmenge, katalogisierte 363
Datenmenge, konsekutiv organisiert 343, 351
Datenmenge, logisch fortlaufend organisiert 343
Datenmenge, nichttemporäre 375
Datenmenge, Organisationsform 465, 471, 474
Datenmenge, permanente 348
Datenmenge, temporäre 348, 361
Datenmenge, untergliedert 304, 361, 806
Datenmenge, virtuell sequentiell organisiert 376, 379, 374, 375, 376, 377, 378, 380, 381, 382, 383, 384
Datenmengenende 359
Datenmengenetikett 313
Datenmengenkennsatz 313, 314
Datenmengenname 341
Datenmodus 902
Datenname 638, 639, 645, 648, 650, 651, 675, 676, 678, 679, 680, 683, 685, 686, 687, 699, 706, 711, 715, 989
Datenname, indizierter 679
Datenparameter 852
Datenprüfung 827, 835
Datenpuffer 948
Datensatz 197, 198, 351, 369, 370, 371, 375, 376, 377, 378, 381, 382, 465, 599, 650, 681, 686, 693, 695, 696, 841, 922
Datenschutz 897
Datenschutzcode 949
Datensichtgerät 98
Datenspezifikation 422
Datensteuerblock 892, 894, 895, 896, 897, 898, 899, 910, 919
Datenteil 369, 370, 371, 372, 377, 633, 634, 644, 645, 646, 647, 649, 651, 654, 655, 657, 659, 665, 669, 674, 679, 684, 685, 686, 687, 802
Datenträger 113, 312, 317, 334, 342, 343, 349, 360, 361, 379, 383, 407, 413, 415, 418, 453, 469, 692, 699, 700, 828, 857, 867, 878, 903, 906, 931, 962
Datenträger-Inhaltsverzeichnis 317, 318, 348
Datenträgerende 698, 862, 902, 903
Datenträgerendeetikett 335
Datenträgerkennsatz 312, 318
Datenträgerkennung 342, 343, 379, 383
Datenträgernummer 342
Datenträgerwechsel 931
Datentyp 390, 521, 522, 667, 676
Datentypumwandlung 443
Datenübertragung 478, 664, 665
Datenumwandlung 407, 415, 445, 463, 528, 529, 530, 531
Datenverarbeitsanlage 44, 45
Datenverarbeitung 324
Datenverarbeitungssystem 44, 57, 100, 632
Datenvereinbarung 536, 674
Datenverlust 113
Datenzeichen 459
Datum 341, 345
DAY 696
DC 789, 792, 798, 809
DCB 362, 892, 894, 897, 898, 899, 902, 903, 911, 912, 917, 924, 926, 929, 930, 931, 932
DCBD 897
DCL 430, 431, 438, 459, 470, 477, 512, 519, 520, 521, 522, 528, 530, 531, 532, 534, 538
DD 358, 380, 383, 384, 464, 466, 467, 470, 487, 488, 489, 514, 537, 643, 651, 691, 693, 694, 699, 700, 903, 904, 920, 934
DD-Statement 363, 365, 894, 896, 948, 985
DD-Steuerstatement 357, 358, 359, 360, 361, 362, 363, 365
DEC 431
DECB 916, 917, 918, 920, 922, 925, 928, 929, 930

DECIMAL 395, 396, 397, 398, 401, 409, 420, 430, 431, 433, 507, 523, 528, 529, 530, 531
DECIMAL-POINT 690
DECLARE 394, 396, 397, 398, 402, 403, 405, 406, 407, 423, 424, 426, 430, 431, 434, 435, 437, 440, 445, 459, 464, 466, 468, 469, 470, 498, 534, 536
DEF 519
DEFAULT 396, 397, 403, 426, 506, 507
DEFINE 379, 380, 384
DEFINE FILE 593, 594, 595
DEFINE STORAGE 778
DEFINED 519, 520, 521
Definierte Konstante 795
Definierte Variable 519, 521
Definition 789, 790, 799, 944
Dekadischer Logarithmus 37, 433
Deklaratives Makro 776
DELETE 383, 384, 486, 487, 488, 489, 490, 942, 945, 946
DEPENDING 681, 707
Dezimalarithmetik 85, 243
Dezimalbruch 529
Dezimaldaten, externe 669
Dezimaldaten, interne 669
Dezimaldatenelement, externes 657, 658, 659, 660, 665, 666, 695, 712, 713
Dezimaldatenelement, internes 659, 660, 665, 666, 712, 713
Dezimale Daten 392
Dezimale Konstante 393, 528, 794
Dezimaler Direktwert 779, 791
Dezimalkomma 21
Dezimalpunkt 21, 23, 89, 252, 300, 301, 302, 303, 388, 392, 395, 396, 458, 460, 462, 546, 561, 577, 638, 639, 646, 657, 658, 661, 662, 664, 666
Dezimalstelle 427
Dezimalsystem 20, 21, 22, 27
Dezimalüberlauf 95, 244
Dezimalzahl 44, 45, 83, 84, 340, 368, 415, 416, 417, 421, 441, 474, 793, 802, 804
Dezimalziffer 388, 394, 458, 459, 460, 471
Diagnostik 91
Dienstprogramm 199, 318, 326, 327, 328, 344, 595, 805, 963, 964
Differenz 650, 672
Dimension 435, 437, 440, 453, 533, 581, 582
DIMENSION-Vereinbarung 581, 582, 584, 585, 588, 597, 598, 599
Dimensionsattribut 520
Dimensionsfestlegung 437
Dimensionsgrenze 441, 508, 520, 532, 533
Dimensionsgrenze, variable 681, 682
DIMOD 869, 874
DIRECT 464, 465, 466, 486, 488, 489, 490
Direkt organisierte Datei 857
Direktbyte 785
Direktdaten 226, 808
Direkte Organisationsform 368, 369
Direkte Verarbeitung 958, 960
Direkte Zugriffsmethode 375, 464, 465, 466, 486, 488, 490, 894
Direkte Zugriffsmöglichkeit 326, 348, 351, 360, 368, 464, 485, 593, 595, 643, 645, 699
Direkteingabe 386
Direkter Zugriff 368, 377, 486, 488, 595, 864, 867, 954
Direktverarbeitung 491
Direktwert 779, 791
Direktwert, dezimaler 779, 791
Direktwert, dualer 779
Direktwert, sedezimaler 779
Disk Label 342
Disk Operating System / Virtual Storage 325
DISP 698, 699, 700
Displacement 60
DISPLAY 414, 640, 641, 646, 647, 655, 656, 660, 661, 682, 685, 695, 696, 706, 707
Disposition 361, 362, 466, 903
Dispositionsoperand 362, 380
Distanzadresse 60, 76, 256, 270, 279, 280, 282, 298, 781, 782, 784, 796, 949
DIVIDE 650, 670
Dividieren 399
DIVISION 690
Division 399, 528, 549, 550, 578, 592, 669, 671, 672, 780, 781
Divisionsfehler 592
Divisionsinstruktionen 66, 86, 219
Divisionsüberlauf 243, 244
DLBL 342, 382, 383, 384, 464, 470, 596
DLOG 551
DO 447, 448, 449,450, 451, 452, 481, 498, 509, 582, 584, 585, 586, 596
DO-Anweisung, äußere 585
DO-Anweisung, ineinandergeschachtelt 585
DO-Anweisung, innere 585

DO-Gruppe 447, 448, 450, 451, 453, 498, 509
DO-Index 452
DO-Nest 585
DO-Schachtelung 585
DOM 938
Doppelfakultät 974
Doppelpunkt 388, 401, 429, 437
Doppelt genaue Konstante 547, 548
Doppelt genaue Variable 548
Doppelte Genauigkeit 216, 217, 222
Doppelwort 53, 54, 83, 90, 114, 216, 219, 221, 222, 247, 248, 249, 277, 278, 393, 400, 522, 524, 547, 548, 549, 550, 646, 665, 939
Doppelwortadresse 53, 810
Doppelwortbegrenzung 936
Doppelwortgrenze 522, 523, 526, 527, 795, 797, 799, 808, 936
Doppelwortlänge 53
DOS 326, 344, 345, 367, 368, 369, 379, 380, 382, 383, 384, 393, 396, 405, 406, 407, 424, 464, 465, 466, 467, 468, 470, 471, 486, 488, 489, 505, 514, 537, 557, 570, 595, 632, 642, 651, 676, 690, 691, 692, 694, 695, 696, 699, 700, 706, 715, 774, 777, 793, 795, 798, 802, 805, 806, 808, 811, 816, 823, 841, 869, 880, 950, 951, 952, 955, 963, 964, 979, 981, 984, 989, 990
DOS/VS 325, 329-345
DOUBLE PRECISION 548, 550, 582, 589
Driftzeichen 462, 463
Druck 880
Druckanweisung 454
Druckaufbereitung 770
Druckausgabe 350, 515, 801, 802, 803, 985
Druckdatei 405, 407, 421, 422, 423, 424, 426, 464, 834, 836
Druckeinstellung 826
Drucken 75, 349, 354, 423, 514, 572, 691, 697, 698, 833, 897
Drucken, wahlweises 802
Drucker 90, 174, 175, 176, 177, 178, 179, 180, 329, 350, 359, 447, 575, 594, 697, 703, 826, 893, 905, 978
Druckerausgabe 576
Druckeroperation 826
Druckfehler 880
Druckformatelement 422
Druckglied 175, 359
Druckkette 175, 359
Druckoption 422
Druckseite 423, 426
Druckstelle 422
Druckzeile 419, 421, 573, 771, 773, 802
DS 778, 798, 799, 809, 813
DSECT 812, 844, 845, 846, 847, 848, 913, 914
DTF 824, 825, 829, 852, 869, 875
DTFCD 824, 825, 830, 831, 832, 833, 834, 841, 870
DTFCN 836, 837, 869, 875
DTFDA 857, 858, 859, 860, 861, 862, 863, 874, 878
DTFDI 855
DTFIS 863, 864, 865, 866, 867, 868, 869
DTFMT 838, 839, 840, 841, 842
DTFPH 877, 878
DTFPR 834, 835, 836, 841, 871
DTPSD 848, 849, 850, 851
Dualbruch 529
Dualdaten 392, 459, 660
Dualdatenelement 660, 665, 712
Duale Daten 392, 459
Duale Konstante 393, 791, 792, 793
Duale Variable 452
Dualer Direktwert 779
Dualkarte 770, 882
Dualkartenverarbeitung 468, 833
Dualpunkt 61, 395, 794
Dualsystem 21, 22
Dualwort 400
Dualzahl 21, 26, 27, 44, 83, 84, 88, 89, 243, 251, 276, 368, 369, 377, 393, 443, 548, 572, 635, 636, 646, 779, 795, 974
Dualziffer 391, 393, 394, 410, 779, 792
Dummy Section 812
DUMP 383, 590, 885, 886
Duplikatsatz 867, 910
Duplikatsatzfehler 863
Duplizieren 6, 8, 9, 10
Durchflußleistung 374, 375
Durchsatzerhöhung 466
Durchsuchen 201, 368, 371, 372, 869
Durchsuchen, erweitertes 368
DV-System 325, 344, 655
DXR 941
Dynamische Adreßumsetzung 264, 265, 266-270
Dynamische Kürze 290
Dynamische Programmverbindung 945, 946
Dynamische Pufferung 919, 925, 926, 928

E-Typ 795, 798
EA 43, 57, 74, 110, 134, 287, 305, 351, 355, 404, 453, 466, 484, 600, 644, 825, 836, 855, 860
EA-Adresse 109, 127
EA-Anweisung 407, 453, 478, 490, 545, 593, 598
EA-Bedingung 426
EA-Block 930, 931
EA-Datei 849, 978
EA-Einheit 574, 593, 691, 876
EA-Einheit, logische 329, 332
EA-Fehler 831, 835, 838, 879, 895, 925, 929
EA-Gebiet 305, 306, 367, 405, 465, 484, 772, 776, 830, 831, 832, 835, 837, 839, 841, 847, 850, 853, 855, 858, 860, 861, 864, 866, 914, 960
EA-Gerät 91, 92, 98, 99, 100, 101, 106, 109, 112, 113, 114, 116, 121, 123, 125, 127, 128, 129, 137, 201, 290, 291, 327, 330, 331, 332, 348, 350, 354, 360, 406, 489, 633, 827, 831, 837, 838
EA-Instruktion 93, 108, 109, 121
EA-Kapitel 690, 691, 693
EA-Operation 74, 107, 109, 111, 112, 114, 117, 120, 124, 128, 327, 344, 351, 358, 367, 419, 484, 500, 577, 823, 829, 845, 853, 892, 893, 894, 921, 930, 958
EA-Puffer 693, 830, 833, 848, 954
EA-Routine 367, 374, 823, 829, 892, 894
EA-Statement 407, 691, 698
EA-Steuerungsparagraph 692, 693
EA-Unterbrechungen 91, 110, 111, 247
EA, datengesteuerte 413, 414
EA, formatbehaftete 593
EA, formatgesteuerte 481
EA, listengesteuerte 404, 405, 406, 407, 414, 415, 530
EA, reihenweise 415, 427, 452, 464, 475, 481, 530
EA, satzweise 464, 467, 471, 472, 473, 477, 478, 479, 522
EA, virtuelle 351, 355
EAB 917
EBCDI-Code 47, 48, 58, 82, 83, 93, 175, 390, 391, 433, 482, 572, 655, 656, 779, 806, 833
ECB 916, 917, 918, 921, 928, 930, 938, 953, 960
Echtzeituhr 91
EDIT 415, 416, 421, 422, 480
Effektive Adresse 60, 88, 108, 249, 251, 268, 272
Eigenmaßnahme 535
Eindimensionaler Bereich 679
Einerkomplement 27, 28
Einfachbedingung 715, 716
Einfache Genauigkeit 216, 217, 222
Einfache Neudefinition 521
Einfache Programmverbindung 943, 944
Einfache Struktur 972
Einfacher Ausdruck 780, 781
Einfaches Neudefinieren 519
Einfaches Statement 426
Einfügungszeichen 662
Eingabe 43, 74, 332, 405, 416, 417, 418, 419, 420, 423, 441, 447, 463, 474, 478, 480, 557, 570, 572, 577, 631, 695, 770, 830, 832, 836, 844, 845, 912
Eingabe-Folgeprüfung 804
Eingabe/Ausgabe 43, 57, 90, 91, 324, 463, 574, 592, 593, 598, 634, 823, 824, 892, 893
Eingabe/Ausgabe-Anweisung 435, 545, 594, 595, 596, 597
Eingabe/Ausgabe-Block 917
Eingabe/Ausgabe-Gebiet 850, 866
Eingabe/Ausgabe-Gerät 57, 98, 414, 557
Eingabe/Ausgabe-Instruktionen 62
Eingabe/Ausgabe-Kapitel 690
Eingabe/Ausgabe-Medium 463
Eingabe/Ausgabe-Operation 404, 410, 557, 559, 823
Eingabe/Ausgabe-Routine 823
Eingabe/Ausgabe-Steuerungssystem 299, 823
Eingabe/Ausgabe-Unterbrechungen 91
Eingabe/Ausgabe, datengesteuerte 413, 414, 415
Eingabe/Ausgabe, formatgesteuerte 414, 415, 416, 417, 418, 419, 420, 421, 453
Eingabe/Ausgabe, listengesteuerte 404, 405, 406, 407, 408, 410, 413, 414
Eingabe/Ausgabe, reihenweise 463, 464
Eingabe/Ausgabe, satzweise 463-470
Eingabe, formatgesteuerte 480
Eingabe, satzweise 469
Eingabeanweisung 332, 439, 583

Eingabedatei 341, 403, 405, 409, 421, 692, 828, 830, 840, 849, 872, 892, 978, 983, 984, 989
Eingabedaten 404, 407, 408, 413, 414, 562, 713
Eingabedatenliste 415
Eingabedatenmenge 358, 987
Eingabedatenreihe 427
Eingabedatenträger 406, 413
Eingabeeinheit 339, 340
Eingabefehler 910
Eingabefeld 415, 416, 417
Eingabeformat-Steuerung 804
Eingabegebiet 475, 699, 837, 853, 868
Eingabegerät 330
Eingabemedium 545, 571, 598
Eingabeoperation 367, 837
Eingabeprozedur 990
Eingabepuffer 478, 833, 846
Eingaberoutine 825
Eingabesatz 572, 599, 695, 855
Eingabestatement 650
Eingabestrom 348, 349, 358, 363, 912
Eingang 971
Eingangsadresse 841
Eingangsleitung 137
Eingangsname 328, 329, 513, 814, 884, 888, 944, 986
Eingangspunkt 499, 513, 945, 946
Eingangspunkt, primärer 513
Eingangspunkt, sekundärer 513
Eingangsstelle 509, 513, 797, 815, 818, 840, 855, 884, 987
Eingangsstelle, sekundäre 514
Eingebetteter Schlüssel 376, 467, 489
Eingefügte Funktion 432, 433, 434, 476, 477, 481, 482, 505, 513, 514, 516, 529, 532, 536
Einheit 557, 592
Einheit, logische 329, 331, 332, 333, 334, 335, 339, 343, 406, 642, 691
Einheit, logische des Programmierers 878, 887
Einheitenname, symbolischer 331
Einheitennummer 591
Einheitsmatrix 35
Einheitsnummer 557, 558, 592
Einheitsstrecke 18
Einlaufblock 498, 499
Einleitendes Programmladen (IPL) 114, 285, 286, 326, 331, 334, 335, 349
Einpunktbetrieb 99, 109, 122, 126, 127, 213
Einschließungsstatement 536, 537
Einstellen 201, 827
Einstellung 367
Eintragung 329, 682, 683, 684, 685, 714
EJECT 802
Elektronische Rechenanlage 43
Element 504, 514, 520, 522, 523, 524, 525, 526, 527, 532, 645, 665, 667, 669, 676, 680, 681, 682, 683, 696, 707
Elementarstufe 683
ELSE 387, 411, 412, 451, 529, 536, 650, 707, 710, 711
ELSE-Klausel 440, 444, 447, 448
Empfangsfeld 654, 656, 658, 660, 661, 662, 666, 670, 671, 672, 676, 678, 991
END 328, 387, 404, 426, 428, 447, 448, 449, 452, 499, 507, 509, 511, 512, 531, 538, 545, 555, 649, 699, 778, 805, 813
END FILE 558, 592
End of Modul 328
END-OF-PAGE 698
Ende 805
Endetikett 312
ENDFILE 403, 404, 408, 426, 473
ENDFL 866
ENDPAGE 426
ENDREQ 950, 958
Endwert 584, 586, 596
Entblocken 823
Entfernen 489
Entformatisieren 593
Entladen 335, 827, 841
Entpacken 81, 82
ENTRY 508, 512, 513, 523, 813, 814, 875
Entry Sequenced Data Set 375
Entscheidungsblock 971
ENV 431
ENVIRONMENT 405, 408, 409, 423, 424, 431, 465, 468, 469, 470, 471, 486, 488, 489, 490, 522, 690
ENVIRONMENT DIVISION 633, 642, 643
EOC 861, 862
EOF 199, 200, 592, 853, 861, 866, 867, 879, 882, 929
EOJ 845, 885
EOP 698
EOV 862, 930, 931
EOV-Kennsatz 335
EQ 355

EQUAL 651, 711
EQUIVALENCE 588
ERASE 950, 958, 962
Ereignissteuerblock 916, 918, 921, 928, 930, 938, 953, 960
ERET 829, 838, 841, 864
Ergebnisfeld 672
Ergibtanweisung 401, 402, 410, 411, 413, 429, 430, 431, 432, 436, 438, 440, 442, 443, 457, 458, 459, 462, 476, 477, 481, 504, 505, 528, 530, 532, 545, 547, 552
Ergibtanweisung, arithmetische 443, 552, 572, 583
Ergibtanweisung, skalare 436, 457
Ergibtstatement 401, 430, 438
Ergibtzeichen 388, 401, 413, 430, 452, 457, 481, 482, 483, 528, 545, 552
Erhöht genaue Konstante 547
Erhöht genauer Wert 561
Erhöhungswert 584, 597
Erkennungsteil 633, 635, 641, 651
Eröffnen 892, 894, 955, 982
Eröffnen von Dateien 423, 424, 469, 470, 649, 650, 698, 699, 825, 841
Eröffnen, explizites 423, 469
Eröffnen, implizites 423, 469
Eröffnungsprozeß 466
Eröffnungszeit 469
ERRBYTE 861, 862
ERROR 481, 514, 515, 670, 671, 674
ERROR-Bedingung 426, 427, 428
Ersatzspur 196, 197, 213
Ersetzungsmodus 902
Erstellung 487, 495, 893
Erstellung von Dateien 487, 495, 893, 963
Erstellungsdatum 341
Erstprogrammierer 970, 971
Erweiterte Genauigkeit 221, 222, 393, 665, 941
Erweiterte mnemonische Operationsschlüssel 786, 787
Erweiterte Steuerung 92, 110, 230, 244, 245, 247, 248, 265
Erweiterte Zugriffsmethode 898, 919
Erweiterter Zugriff 367
Erweitertes Durchsuchen 368
ESD 328, 329
ESDS 375
ESETL 868, 869, 909
Etikett 406, 468, 643, 645, 825
Etikettyp 362
EVENT 523
Event Control Block 916
EVERY 707
EXCLUSIVE 465
Exclusive Or 25, 26
EXCP 879, 881, 931
EXEC 336, 337, 338, 339, 340, 356, 357, 383, 384, 516, 537, 538, 652, 819, 887, 934, 943
EXEC-Statement 357, 364
EXEC-Steuerstatement 336, 337, 338, 342, 356, 357, 358, 363
EXHIBIT 706, 707
EXIT 515, 591, 886
Exklusives Oder 25, 26, 227
EXLST 950, 951, 952, 955, 956, 961
EXP 551
Explizite Adresse 784
Explizite Typvereinbarung 548, 552, 582, 590
Explizites Abschließen 423
Explizites Eröffnen 423, 469
Exponent 25, 216, 222, 393, 399, 400, 401, 417, 432, 460, 461, 546, 547, 550, 560, 562, 638, 664, 672, 794
Exponentenfeld 460
Exponentenform 25
Exponentenüberlauf 222, 244, 427
Exponentenunterlauf 95, 222, 224, 428
EXTENT 342, 343, 380, 382, 383, 464, 850, 858, 862, 878
EXTENTNUMBER 468
EXTENT-Statement 343, 344
EXTENT-Steuerstatement 344
EXTERNAL 396, 398, 405, 465, 469, 477, 499, 500, 501, 503, 521
External Symbol Dictionary 328
Externe Bezeichnung 389, 390, 428
Externe Dezimaldaten 669
Externe Gleitpunktdaten 664, 665, 669
Externe Prozedur 500, 509, 510, 512
Externe Unterbrechung 90, 91, 238, 248, 276, 885, 886
Externer Gleitpunktwert 696
Externer Name 398, 399, 499, 505, 512, 813, 814
Externer Speicher 266, 271
Externes Dezimaldatenelement 657, 658, 659, 660, 665, 666, 695, 712, 713
Externes Feld 416
Externes Gleitpunktdatenelement 664, 665, 695, 712
Externes Medium 415, 416, 417, 427, 435, 463
Externes Signal 247, 248
Externes Speichergerät 351, 633, 643, 645, 824

Externes Speichermedium 419, 420
Externes Symbol 814
Externes Symbolverzeichnis 328, 329
Externspeicher 415
EXTRN 814

F-Format 897, 903, 921
F-Typ 794, 795, 798
Fachauswahl 466, 832, 833, 893
Fakultät 509
Falsch 409, 711, 713
Falsche Satzlänge 105, 106, 114, 117, 119, 202, 829, 861, 862, 867, 925
FCOBOL 336
FD 644, 645, 647, 649, 674
Fehlerbedingung 823, 824, 879, 911
Fehlerbehandlung 823, 824, 829, 961
Fehlerbeseitigung 917
Fehlerbyte 861
Fehlermeldung 869
Fehlernachricht 335, 562
Fehlerschlüssel 858
Feld 3, 7, 8, 53, 54, 79, 81, 82, 85, 86, 87, 89, 199, 302, 376, 395, 396, 415, 416, 417, 419, 421, 427, 428, 458, 460, 461, 462, 471, 560, 561, 635, 636, 648, 651, 658, 659, 661, 671, 676, 702, 779, 789
Feld, aufbereitetes 462
Feld, externes 416
Feld, numerisches 458, 460, 571
Feldbreite 415, 418, 419
Feldform 521
Feldlänge 415, 417, 419, 578
FEOV 828, 877, 878, 920
FEOVD 828, 850, 858
Feste Länge 369, 486, 487, 488, 645, 872, 873, 903
Festkommadarstellung 61
Festkommadaten 61
Festpunktdarstellung 61, 94, 400, 435, 452, 458, 462, 474, 499, 665, 793, 794, 819, 980
Festpunktdaten 61, 236, 277, 392, 395
Festpunktdatenelement, numerisches 707
Festpunktdivision 528
Festpunktinstruktionen 61
Festpunktkonstante 393
Festpunktliteral 638, 665, 666, 669
Festpunktrechnung 428
Festpunktüberlauf 94
Festpunktvariable 398
Festpunktzahl 61, 62, 394
FETCH 884
FFORTRAN 336
Figürliche Konstante 639, 683
Figurative Konstante 639, 647, 648, 650, 651, 665, 666, 667, 683, 696, 709, 711, 712
FILE 379, 380, 381, 382, 383, 405, 407, 413, 415, 423, 464, 469, 470, 481, 484, 485, 486, 488, 490, 523, 532, 644, 693, 694
FILE-CONTROL 692
FILE-LIMIT 692, 693
FILE-LIMITS 692, 693
FILLER 645, 680
FIND 595
FINISH 428, 515
FIXED 395, 396, 397, 398, 401, 409, 420, 427, 430, 437, 443, 458, 506, 507, 520, 523, 528, 529, 530, 531, 532
FIXEDOVERFLOW 427, 428, 431, 528
FLOAT 395, 396, 397, 398, 401, 409, 420, 433, 507, 523, 530
FOFL 431
Folgekarte 804
Folgeprüfung 804
Folgespalte 774
Folgestanzer 1
Folgezeile 543, 545
FOR 381, 692, 693
Format 348, 369, 374, 486, 488, 522, 557, 562
FORMAT 419, 422, 426, 545, 547, 552, 557, 559, 560, 561, 562, 563, 570, 571, 572, 573, 574, 575, 576, 595, 596, 698, 600
Formatangabe 559, 560, 561, 562, 563, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 599
Formatangabe, alphanumerische 570, 571
Formatangabe, Gruppe 575
Formatangabe, vielfache 573
Formatart 368, 369
Formatbehaftete EA 593
Formatelement 415, 416, 417, 418, 419, 420, 421, 422, 426, 427, 453, 454, 480, 481
Formatfrei 593, 594
Formatfreies Lesen 593, 594
Formatfreies Schreiben 593, 594
Formatgesteuert 463, 481
Formatgesteuerte Ausgabe 422
Formatgesteuerte Ausgabeanweisung 415

Formatgesteuerte EA 481
Formatgesteuerte Eingabe 480
Formatgesteuerte Eingabe/Ausgabe 414, 415, 416, 417, 418, 419, 420, 421, 453
Formatisieren 593
Formatliste 414, 415, 418, 419, 421, 422, 453, 454, 480
Formel 449
Formular 359, 697
Formularsteuerpuffer 170, 359
Formularsteuerung 691, 697
Formularsteuerungspuffer 176, 359
Formulartransport 421, 422, 691
Formularvorschub 697
Fortlaufender Zugriff 367, 864
FORTLIB 347
FORTRAN 324, 325, 326, 336, 364, 367, 369, 376, 510, 522, 542, 543, 545, 555, 557, 570, 587, 595, 596, 631, 665, 770, 819
FORTRAN IV 542
FORTRAN-Programm 555, 558, 581
FORTRAN-Sprache 542, 543, 545, 557, 558, 561, 563, 570, 575, 581, 589, 592, 593, 598
FORTRAN-Statement 543, 545, 547
Fortsetzungskarte 352
Fortsetzungszeichen 380
Fortsetzungszeile 675, 774, 776
Fragezeichen 388
FREE 534, 535, 536
FREEBUF 928
FREEDBUF 919, 925
FREEMAIN 937
FREEPOOL 899, 900, 901, 911, 919, 924, 928
FREESPACE 381, 382
FREEVIS 888
Freigabebefehl 356
Freigabestatement 535
Freispeicher 376, 377, 378
FROM 469, 478, 484, 485, 488, 490, 522, 649, 650, 669, 670, 695, 696, 697, 699
Führende Null 368, 392, 415, 416, 460, 462, 487, 562, 578, 664
Führende Ziffer 462
Füllbit 522, 791
Füllbyte 522, 526, 527, 682
Füllzeichen 254
Funktion 432, 433, 434, 437, 481, 505, 508, 509, 510, 513, 514, 516, 549, 550, 987
Funktion, eingefügte 432, 433, 434, 476, 477, 481, 482, 505, 513, 514, 516, 529, 532, 536
Funktion, geschlossene 551
Funktion, offene 551
Funktionsdefinition 505
Funktionsname 432, 438, 505, 506, 507, 508, 509, 691, 697, 715
Funktionsprozedur 505, 506, 507, 508, 509, 512
Funktionsprozedur, rekursive 510
Funktionsunterprogramm 514, 819
Funktionswert 300, 432, 433, 434, 476, 506, 507, 508, 509, 510, 513, 516, 536, 551, 974

Ganze Zahlen 18, 37
Ganzzahlige Konstante 546, 557, 583
Ganzzahlige Variable 547, 548, 549, 579, 590, 591, 592, 596
Gattungsname 350
Gattungsschlüssel 374, 376
Gaussche Verteilung 31
Gaussche Zahlenebene 40, 41
GE 355
Gebiet 533, 534, 535, 829, 832, 834, 836, 840, 842, 844, 845, 846, 852
Gebietslänge 864
Gebietsumfang 534
Gebietsvariable 534
Geblockter Satz 367, 374, 375, 467, 527, 838, 841, 842, 850, 851
Gekennzeichneter Name 458, 685, 686
Gemeinsamer Abschnitt 328, 810, 811
Gemeinsamer Steuerabschnitt 328, 810, 811
Genauigkeit 392, 393, 394, 395, 398, 400, 433, 443, 462, 528, 530
Genauigkeit, doppelte 216, 217, 222
Genauigkeit, einfache 216, 217, 222
Genauigkeit, erweiterte 221, 222, 393, 665, 941
Genauigkeitsattribut 395, 396, 397, 529
Generation 304, 316, 319, 534, 536
Generationsnummer 316, 319, 342
Generischer Schlüssel 466, 488, 489, 909, 954
GENKEY 466, 488, 489, 490
Gepackte Daten 81
Gepufferter Zugriff 367
Geradengleichung 32
Gerät, virtuelles 351
Geräteabhängigkeit 643
Geräteadresse 112, 114, 197
Geräteanzeigeleiste 330
Geräteende 826, 879, 916, 931

Gerätename 695, 696, 835, 837, 838, 855
Geräteprüfbyte 138, 139, 209-212
Gerätestatusbyte 107, 123, 124, 127, 129, 137, 138, 209-212
Gerätesteuerzeichen 466, 831, 835
Gerätetyp 329, 350, 367, 405, 406, 467, 643, 832, 835, 850, 858, 864, 895, 898
Gerätzuweisung 331, 332
Geschichtliche Entwicklung der Lochkartentechnik 1
Geschlossene Funktion 551
Gesetzter Sprung 589
Gesteuerte Speicherklasse 535, 536
Gestreut organisierte Datei 857
Gestreut organisierte Datenmenge 892, 894
Gestreute Organisationsform 894
GET 387, 405, 406, 407, 408, 409, 413, 414, 415, 416, 418, 419, 423, 427, 436, 452, 469, 479, 480, 481, 483, 823, 825, 826, 829, 832, 833, 834, 836, 837, 840, 846, 848, 849, 851, 852, 865, 866, 868, 893, 894, 904, 905, 906, 907, 908, 910, 950, 954, 958, 962
GETBUF 919, 921, 924, 928
GETIME 886
GETMAIN 936
GETPOOL 899, 900, 919, 924, 936
GETVIS 337, 339, 887, 888
Gewöhnlicher Logarithmus 37
GIVING 670
Gleich 787
Gleichheitszahlen 637, 640
Gleichheitszeichen 352, 387, 388, 545, 552, 779, 797
Gleichung 28, 32
Gleichung, lineare 435
Gleichung, quadratische 447, 448
Gleichungssystem 28, 32, 35
Gleitkommadarstellung 23, 216, 244
Gleitpunktarithmetik 941
Gleitpunktdarstellung 23, 25, 28, 216, 217, 218, 219, 244, 416, 434, 458, 460, 461, 514, 664, 795, 941, 980, 983
Gleitpunktdaten 217, 219, 392, 395, 396, 665
Gleitpunktdaten, externe 664, 665, 669
Gleitpunktdaten, interne 665, 669
Gleitpunktdatenelement, externes 664, 665, 695, 712
Gleitpunktdatenelement, internes 665, 712
Gleitpunktform 416
Gleitpunktinstruktionen 217, 218
Gleitpunktkonstante 393, 794, 795
Gleitpunktliteral 638, 665, 666, 669
Gleitpunktrechnung 427, 428
Gleitpunktregister 35, 217, 220, 221, 783, 819
Gleitpunktüberlauf 591
Gleitpunktunterlauf 591
Gleitpunktvariable 398
Gleitpunktwert 433, 819
Gleitpunktwert, externer 696
Gleitpunktzahl 392, 393, 394, 396, 400, 546, 548, 665, 795
Gleitzeichen 663
Glied 361, 399, 400, 537, 778, 779, 780, 781, 798, 806
Glied, selbst definierendes 779
Gliedname 361
GO 387, 403
GO TO 554, 583, 585, 587, 588, 589, 650, 675, 707, 708
GOBACK 709
GOTO 403, 404, 412, 430, 451, 499, 509, 513, 515
GR-Verfahren 182, 184
Gradmaß von Winkeln 39
GREATER 651, 711
Greenwicher Zeit 887
Grenze, obere 437, 533
Grenze, untere 437
Größer-als-Zeichen 388, 534
Großbuchstabe 359, 368, 398, 637, 640, 827
Großcomputer 386
Grundteil 674
Gruppe 429, 447, 448, 449, 450, 498, 499, 645, 666, 667, 676, 679, 680, 683, 684, 685, 686
Gruppe von Formatangaben 575
Gruppenaufzeichnungsverfahren 182
Gruppenname 351
GT 355
Gültigkeitsbereich von Namen 396, 498, 499, 509
Gültigkeitsbereich 498, 499, 500, 509
Gültigkeitsbereichattribut 501
Gültigkeitsfehler 880
Gültigkeitszeichen 465

H-Typ 794, 795, 798
Hängende Unterbrechung 110, 120, 121, 126, 128

Halbbyte 54, 80, 82, 83, 87, 233, 302, 392
Halbwort 53, 54, 61, 94, 224, 249, 376, 377, 524, 532, 548, 549, 635, 646, 819, 852
Halbwortadresse 53
Halbwortgrenze 523, 794, 796, 798
Halbwortinstruktionen 224
Halbwortlänge 53
Haltewarteschlange 356
Hardware 326, 463, 465
Haufen 378, 379, 380, 381, 382, 383
Hauptindex 343, 373, 375, 379, 489, 865, 897
Hauptinhaltsverzeichnis 373, 466, 468, 488, 489, 865, 867
Hauptkatalog 347, 348, 384
Hauptprogramm 111, 299, 505, 506, 507, 508, 509, 511, 512, 514, 537, 538, 796, 812, 813, 816, 819, 943, 973, 974
Hauptprozedur 510
Hauptsegment 512
Hauptspeicher 52, 54, 56, 59, 62, 88, 93, 94, 95, 98, 103, 105, 109, 114, 118, 120, 201, 202, 206, 224, 226, 231, 241, 242, 247, 248, 252, 266, 267, 271, 274, 278, 325, 326, 327, 348, 371, 372, 374, 375, 396, 420, 427, 431, 438, 463, 466, 489, 495, 515, 533, 534, 535, 545, 547, 552, 561, 578, 588, 590, 593, 671, 693, 703, 776, 798, 799, 851, 852, 864, 884, 885, 900, 934, 945
Hauptspeicheradresse 772, 774
Hauptspeicherausrichtung 523
Hauptspeicherauszug 590
Hauptspeicherbelegung 791, 792, 794
Hauptspeicherblock 235, 242, 273
Hauptspeichergebiet 478, 488, 489, 936
Hauptspeicherklasse 500, 501, 502, 503, 504, 505
Hauptspeicherplatz 476, 477, 500, 501, 515, 521, 522, 532, 533, 534, 581, 590, 694, 772, 777, 790, 797, 845
Hauptspeicherschutzeinrichtung 238
Hauptspeicherunterteilung 522
Hauptspeicherzyklus 52
Hauptstruktur 455, 456, 458, 500, 504
Hauptstruktur, nichtindiziert 469
Hauptstrukturname 476
Hauptstrukturteil 524
Hauptwert 432
Hausadresse 198, 200, 318
Head 368
HEX 383
Hexadezimalsystem 21
HIGH 433
HIGH-VALUE 66, 639
HIGH-VALUES 639
HIGHINDEX 468
Hilfsgerät 643
Hilfsspeicher 849
Hinweisoperation 367
Hinzufügen 893
Hochkomma 406
Höhere Rangordnung 444
Hollerith, Hermann 1, 2
Hollerithangabe 570
Hollerithcode 2, 13, 14, 58
Hollerithfeld 571

I-O-CONTROL 692
IBM 1, 13, 324, 325, 359, 363, 364, 386, 538, 557, 676, 805, 819
IBM /360 325, 374, 632, 637, 643, 655, 673, 690
IBM /370 323, 329, 374, 386, 392, 393, 394, 395, 409, 414, 427, 435, 455, 542, 543, 545, 546, 570, 571, 632, 635, 637, 643, 655, 673, 690, 770, 811, 823, 824
IBM 129 9, 10, 11, 14, 15
IBM 1316 194, 195, 212
IBM 1403 174, 175, 176, 177, 178, 179, 180, 332, 350, 351, 643, 835
IBM 1442 332, 333, 643, 827, 832
IBM 1443 174, 175, 176, 177, 178, 179, 180, 332, 351, 594, 643
IBM 1445 643
IBM 2303 213, 214
IBM 2305 213, 350, 643
IBM 2311 212, 213, 214, 333, 468, 643, 850, 858, 864
IBM 2312 212
IBM 2313 212
IBM 2314 212, 213, 214, 333, 350, 406, 468, 643, 850, 858, 864
IBM 2316 194, 195, 212
IBM 2319 212, 333, 350
IBM 2321 850, 858, 859, 864
IBM 2400 333, 350
IBM 2401 192, 193, 194
IBM 2402 192, 193, 194
IBM 2403 192, 193, 194
IBM 2404 192, 193, 194
IBM 2415 192, 193, 194
IBM 2420 192, 193, 194

IBM 2501 139, 140, 141, 142, 143, 332, 351, 406, 643, 870
IBM 2520 103, 104, 143, 144, 145, 146, 332, 333, 351, 643, 827, 832, 870
IBM 2540 146, 147, 148, 149, 150, 151, 152, 332, 333, 351, 406, 825, 827, 832, 870
IBM 2560 332, 333, 827, 831, 835
IBM 2596 161, 162, 163, 164, 332, 333, 870
IBM 2803 192, 194
IBM 2804 192, 194
IBM 2835 213
IBM 2841 212, 213
IBM 2844 212
IBM 29 3, 4, 9, 10, 11
IBM 3125 170-174
IBM 2303 175, 176, 177, 178, 179, 180, 332
IBM 3210 836
IBM 3211 175, 176, 177, 178, 179, 180, 332, 351, 359, 643, 827, 835, 880
IBM 3215 168, 169, 170, 836
IBM 3330 212, 213, 214, 333, 343, 344, 350, 369, 406, 643, 850, 858, 864
IBM 3333 212, 213, 214, 333, 350
IBM 3336 194, 195, 198, 212, 213, 214
IBM 3340 213, 333, 350, 643, 850, 858, 864
IBM 3348 213
IBM 3400 350
IBM 3410 192, 194, 333
IBM 3411 192, 194
IBM 3420 185, 186, 187, 188, 189, 190, 191, 192, 193, 333, 406
IBM 3504 153, 154, 155, 156, 157, 158, 332, 827, 831, 870
IBM 3505 153, 154, 155, 156, 157, 158, 332, 351, 406, 467, 468, 643, 827, 831, 870
IBM 3525 159, 160,161, 332, 333, 351, 423, 467, 468, 643, 827, 831, 835, 836, 870, 906
IBM 3803 192
IBM 3830 213
IBM 5203 175, 176, 177, 178, 179, 180, 332, 835
IBM 5425 164, 165, 166, 167, 332, 333, 827, 831, 835
IBM 5496 14, 15
IBM-Prozedur, standardisierte 365
ICTL 804
ID 641
IDAW 105, 118, 120
IDCAMS 379, 380, 383, 384
IF 387, 411, 412, 426, 429, 440, 444, 447, 448, 477, 498, 529, 536, 554, 572, 583, 584, 585, 650, 651, 684, 710, 711, 714
IF-Nest 711
IGNORE 469, 485, 488, 490
ILS 94
IMAGE 482
IMAGELIB 347, 359
Imaginärteil 40, 41, 394, 395, 400, 401, 418, 459, 482
Imaginärzahl 40
Imperativanweisung 670, 671, 698, 699, 707, 710
Imperatives Makro 776
IMPLICIT 589, 590
Implizite Adresse 784
Implizite Länge 786, 790, 796, 798, 810
Implizites Abschließen 423, 470, 473
Implizites Eröffnen 423, 469
IN 534, 535, 536, 685, 686
INCLUDE 537
INDENTIFICATION DIVISION 633, 641
Indentifikationsnummer 938
Index 19, 20, 60, 343, 369, 434, 436, 437, 438, 439, 441, 456, 458, 514, 520, 581, 582, 588, 596, 679, 680, 681
INDEXAREA 488
Indexausdruck 435
INDEXED 381, 465, 466, 467, 488, 490
Indexkomponente 948, 949
INDEXMULTIPLE 468, 489
Indexpunkt 195, 197, 198
Indexregister 60, 270
Indexsatz 370, 371, 372, 382
Indexschreibweise 19, 435
Indexsequential Acces Method 376
Indexsequentiell organisierte Datei 863, 882
Indexsequentiell organisierte Datenmenge 361, 376, 892, 894, 897
Indexsequentiell organisierter Datenbestand 343
Indexsequentielle Organisationsform 321, 338, 342, 343, 369, 370, 371, 372, 373, 374, 375, 376, 379, 466, 467, 468, 488, 489, 490, 491, 494, 894
Indexstufe 949
Indirekte Adressierung 105, 120
Indirektes Datenadreßwort 105, 118, 120, 273, 279
Indizierte Adresse 60
Indizierte Variable 437, 451, 581

Indizierter Datenname 679
Indizierung 596
Induktive Schleife 583, 584, 585
Ineinandergeschachtelte DO-Anweisung 585
INFILE 383
Infixoperator 399
Informationsaustausch 345, 883, 887, 937
Informationsaustausch, Bereich 883, 887
Informationsaustauschgebiet 345
Inhaltsverzeichnis 343, 361, 369, 373, 374, 864
INIT 431, 520
INITIAL 431, 438, 441, 477
Initial Program Loading 326
Initialisierung 297, 431, 438, 439, 440
Initiator 348, 349, 350
Inkrement 584, 586, 597
Innere DO-Anweisung 585
INPUT 405, 423, 464, 465, 469, 485, 486, 488, 489, 490, 649, 698, 699
INPUT-OUTPUT 642, 690, 692
Input/Output Block 917
Input/Output Control System 823
Instruktion 42, 56, 59, 60, 61, 81, 82, 87, 88, 89, 99, 108, 109, 219, 220, 224, 226, 230, 232, 249, 250, 257, 297, 323, 328, 400, 406, 410, 507, 508, 522, 551, 553, 581, 584, 768, 769, 770, 771, 772, 776, 777, 782, 783, 786, 787, 797, 806, 823, 825, 828, 860, 873, 885, 888, 889, 941, 944, 976
Instruktion, arithmetische 545, 547, 552
Instruktion, logische 545, 547, 552
Instruktion, symbolische 776
Instruktion, untergeordnete 257, 258
Instruktionsfolge 56, 69, 133, 231, 232, 249, 323, 451, 683, 783, 823, 945
Instruktionsformat 59, 61, 62, 260, 261
Instruktionslängenschlüssel 94, 230, 241, 242, 917
Instruktionsmenge 290, 297
INTEGER 548, 550, 582, 584, 589, 590
Interface 121, 123, 125, 127, 128, 129, 137
Interleaved Mode 99
INTERNAL 396, 398, 405, 465, 469, 500, 501, 503, 505, 521
International Business Machines Corporation 1
Interne Bezeichnung 389, 511
Interne Dezimaldaten 669
Interne Gleitpunktdaten 665, 669
Interne Prozedur 508, 509, 510, 512
Interne Subroutineprozedur 512
Interner Block 500
Interner Name 398, 505, 813
Internes Dezimaldatenelement 659, 660, 665, 666, 712, 713
Internes Gleitpunktdatenelement 665, 712
Internes Register 55
Internspeicher 52
Interpunktionsregel 639, 640, 674, 680
Intervallzeitgeber 247, 274, 275, 278, 885, 886
INTO 469, 478, 484, 485, 486, 488, 490, 522, 649, 650, 670
Inverse Matrix 35, 36
IOB 916, 917, 930, 931
IOCS 299, 823, 824, 825, 826, 834, 838, 841, 847, 862, 863, 869, 870, 871, 872, 873, 874, 875, 876, 877, 879, 899, 929, 930, 931
IOCS-Routine 824, 825, 830
IOCS, logisches 824, 828, 830, 882, 894
IOCS, physikalisches 894
IOCS, physisches 824, 876, 877, 878, 879, 880, 881, 882, 883, 930, 931, 932, 933, 934
IPL 114, 285, 286, 326, 334, 349,
IPL-Lader 326
IPT 329
IS 644, 647, 651, 685, 690, 692, 713, 714, 715
ISAM 376, 898
ISC 342
ISE 342
ISEQ 804
ISMOD 869, 874, 875
ISS 896
iSUB 520, 521
Iterative Schleife 450

JCL 341, 537
JDUMP 885, 886
Jederzeit wiederbetretbar 973
JOB 336, 338, 339, 352, 353, 652, 943
Job 327, 333, 334, 336, 339, 340, 341, 344, 345, 348, 349, 351, 353, 354, 355, 356, 357, 360, 361, 380, 478, 516, 558, 671, 882, 885, 904, 912
Job Control Language 341
JOB-Statement 357, 384, 676

JOB-Steuerstatement 336, 345, 351, 352, 353, 354, 355, 356, 357
Jobausführung 356
JOBCAT 358, 384
Jobende 75, 336, 339, 345, 361, 650, 826, 886, 911
Jobendestatement 336
Jobklasse 349, 350, 354, 355
JOBLIB 358, 946, 952
Jobname 336, 340, 352, 356, 652
Jobstep 336, 339, 345, 346, 349, 354, 355, 356, 357, 358, 360, 361, 363, 558, 596, 676, 884, 885, 887
Jobstepende 361
Jobsteuersprache 341, 345, 356, 380, 382, 383, 384, 386, 405, 463, 466, 467, 470, 487, 488, 489, 514, 516, 537, 595, 643, 676, 691, 693, 694, 699, 700, 715, 819, 859, 887, 934, 937, 983
Jobsteuerstatement 331, 335, 336, 337, 348, 349, 352, 354, 356, 984
Jobsteuerung 327, 335, 336
Jobsteuerungsprogramm 327
Jobstop 934, 940, 943, 945
Jobwarteschlange 347
Jobzeit 357
Journalführung 951
Juniorprogrammierer 970
JUST 682
JUSTIFIED 682

Kanal 98, 99, 106, 108, 109, 110, 111, 112, 114, 120, 124, 125, 126, 127, 128, 129, 135, 137, 332, 906, 907
Kanal 1 691, 697, 826
Kanal 9 926, 881
Kanal 12 691, 698, 826
Kanaladresse 100, 110, 114
Kanaladresswort 107, 109, 114, 123, 130, 131, 132, 133, 238, 824, 894
Kanalanzeigeleiste 108, 135
Kanalbedingung 826
Kanalbefehl 98, 99, 118
Kanalbefehlschlüssel 116, 117
Kanalbefehlswort 104, 105, 109, 110, 114, 115, 118, 122, 130, 137, 201, 271, 279, 799, 800, 823, 877, 881, 894, 916, 917, 931
Kanaldatenfehler 137
Kanalende 877, 879
Kanalgeräteadresse 344, 350, 360
Kanalgeräteschleinadresse 351, 355
Kanalidentifizierung 134, 274
Kanalprogramm 103, 104, 105, 106, 108, 109, 111, 116, 123, 130, 374, 375, 824, 876, 916, 917, 930, 931
Kanalstatusbyte 107, 117, 119, 123, 125, 128, 138
Kanalstatuswort 107, 110, 112, 113, 117, 122, 125, 279, 877, 917
Kapitel 674, 684, 706, 708
Kapitelname 642, 674, 687, 703
Karte 386,407, 474, 475, 631, 634, 636, 696, 701, 827, 830, 844, 845
Kartenablage 3
Kartenart 328,342, 474, 475, 476
Kartenauslauf 6, 8
Kartencode 2
Karteneinteilung 2
Kartenlesegerät 336
Kartenleser 57, 99, 102, 103, 106, 117, 344, 351, 691, 906
Kartenlocher 14, 15
Kartenpaket 7, 16, 844
Kartenprüfer 14, 15
Kartensatz 7, 16
Kartensortierer 15
Kartenstanzer 57, 102, 351
Kartesisches Koordinatensystem 38, 39
Katalog 331, 348, 380, 383, 384
Katalogeintragung 361
Katalogisierte Datenmenge 363
Katalogisierte Prozedur 354, 357, 363, 364, 365, 538
Kehrmatrix 35, 36
Kenninformation 514, 515
Kennsatz 312, 313, 314, 315, 316, 317, 321, 340, 348, 362, 406, 468, 643, 645, 692, 823, 828, 838, 839, 841, 852, 859, 877, 892, 950, 982, 983, 990
Kennsatz, Standard- 313
Kennsatzoperand 362
Kennsatzsteuerstatement 338
Kennsatztyp 338, 362
Kennung 379, 383
Kennungsleitung 137
Kennzeichenbyte 877, 882
Kennzeichnung 206, 685, 686, 687, 774, 853, 858, 860, 861, 867, 868
Kennzeichnungsfeld 199, 200, 861, 862, 880
Kennzeichnungsstufe 685
Kette 391, 392, 409, 427, 438, 442, 481, 528, 531, 532, 533, 961
Kettendaten 392, 395, 399

Kettenlänge 531
Kettentyp 451, 482
Kettenvariable 395, 481, 531, 532
Kettenwiederholungsfaktor 439
Kettenzeichen 439
KEY 486, 487, 488, 489, 490
Key sequenced data set 375
KEY-Bedingung 495
KEYED 464, 465, 486, 488
KEYFROM 485, 486, 487, 488
KEYLEN 487
KEYLENGTH 467, 486
KEYLOC 467, 488
KEYS 381, 382
KEYTO 485, 486, 487, 488, 489, 490
Klammer 331, 332, 333, 353, 355, 356, 357, 361, 382, 383, 388, 391, 398, 399, 400, 403, 411, 414, 418, 429, 431, 433, 434, 438, 444, 450, 452, 453, 454, 461, 465, 476, 505, 507, 520, 550, 551, 554, 575, 589, 637, 638, 640, 680, 716, 791
Klammernpaar 575, 576
Klammernstufe 576, 780
Klarschriftleser 102
Klasse 713
Klasse, alphabetische 713
Klasse, nichtalphabetische 713
Klasse, nichtnumerische 713
Klasse, numerische 713
Klassen 31
Klassenbedingung 713
Klassenprüfung 713
Klausel 411, 412, 529, 536, 641, 644, 645, 646, 647, 649, 656, 665, 671, 674, 676, 678, 679, 682, 684, 685, 686, 690, 691, 692, 693, 694, 695, 697, 698, 711
Kleinbuchstabe 359, 398, 827
Kleiner-als-Zeichen 388
Kluft 183, 201, 305, 308, 827, 854
Koeffizient 434,447
Koeffizientenmatrix 33
Kombinierte Datei 830, 831, 832, 833
Komma 336, 337, 352, 353, 357, 380, 388, 392, 401, 404, 406, 413, 435, 438, 452, 460, 483, 505, 637, 662, 674, 680, 691, 779
Kommentar 332,336, 388, 390, 391, 426, 427, 428, 430, 543, 641, 645, 675, 690, 692, 693, 694, 776, 777
Kommentarzeile 675
Kommutatives Gesetz 26
Kompatibilität 670
Kompilier 478
Kompilierer 324, 325, 326, 331, 355, 387, 390, 391, 392, 393, 394, 395, 396, 397, 400, 404, 406, 410, 414, 430, 432, 435, 451, 457, 466, 471, 484, 501, 505, 507, 508, 522, 537, 542, 543, 545, 548, 550, 551, 553, 555, 557, 560, 570, 591, 595, 597, 632, 635, 637, 640, 644, 645, 654, 655, 658, 659, 660, 663, 667, 671, 673, 675, 676, 677, 683, 685, 690, 693, 694, 695, 696, 706, 770
Kompilierung 386
Komplementärzahl 27
Komplementbildung 27, 72
Komplementierung 27, 72
Komplexe Daten 392, 418, 458
Komplexe Konstante 394
Komplexe Variable 482
Komplexe Zahl 40, 41, 395, 418, 433
Komplexer verschieblicher Ausdruck 781, 795
Komplexes Argument 432
Konjugiert Komplexe Zahl 41
Konsekutiv organisierte Datei 592, 595, 643, 698, 857
Konsekutiv organisierte Datenmenge 343, 351
Konsekutive Organisationsform 313, 338, 342, 343, 367, 374, 465, 466, 471, 474, 490, 643, 698, 894
Konsolausgabe 978
Konsole 92, 168-174, 248, 285, 286, 287, 330, 414, 836
Konstante 328, 393, 394, 395, 396, 397, 398, 399, 408, 409, 435, 439, 455, 495, 508, 528, 529, 530, 546, 547, 549, 599, 666, 774, 776, 777, 778, 779, 789, 790, 791, 792, 793, 794, 795, 798, 803, 807, 813
Konstante, arithmetische 529
Konstante, definierte 795
Konstante, dezimale 393, 528, 794
Konstante, doppelt genaue 547, 548
Konstante, duale 393, 791, 792, 793
Konstante, erhöht genaue 547
Konstante, figürliche 639, 683
Konstante, figurative 639, 647, 648, 650, 651, 665, 666, 667, 683, 696, 709, 711, 712
Konstante, ganzzahlige 546, 557, 583
Konstante, komplexe 394
Konstante, numerische 793
Konstante, reelle 546, 547

Konstante, sedezimale 791, 792, 793
Konstante, vorgesehene 639, 683
Konstantentyp 401, 790, 791
Kontrollbereich 949
Kontrollfeld 308
Kontrollgebiet 378, 379, 381
Kontrollinformationsbereich 376, 378
Kontrollintervall 376, 377, 378, 379, 381, 382, 948, 949
Kontrollintervalldefinitionsfeld 376, 377
Konvertierung 21, 83, 84, 573, 794
Koordinatensystem 38, 39
Koordinatenursprung 38, 39
Kopf 857, 867
Kopfstatement 976
Kopfvereinbarung 500, 509
Kopfzeile 802
Kopieren 805
Korrelationskoeffizient 32
Korrespondierende Neudefinition 521
KSDS 375
Kubikwurzel 449, 450

L-modus 902, 904, 907, 909, 911, 913, 922
L-typ 795
LABEL 440, 441, 451, 523, 644, 645
LABEL-operand 362
Ladeinstruktion 797
Ladeinstruktionen 63, 218, 221, 225, 235, 237
Lademodul 328, 329, 330, 336, 340, 365, 973
Lademodus 893, 907, 910, 920, 945
Laden 329, 431, 500, 884, 982
Laden, sequentielles 864
Ladepunkt 183, 405, 840, 841, 904, 916
Lader 364, 365, 538
Länge 395, 779, 790, 791
Länge, feste 369, 486, 487, 488, 645, 872, 873, 903
Länge, implizite 786, 790, 796, 798, 810
Länge, unbestimmt 903
Länge, undefiniert 487, 645, 650, 833, 859, 872
Länge, variable 467, 489, 531, 532, 533, 645, 650, 840, 872, 903, 925, 988
Längenangabe 785, 786, 792, 794, 795, 798, 799
Längenattribut 395, 414, 772, 809
Längenfaktor 791
Längenoperand 785, 791
Längenschlüssel 79, 81
Längenspezifikation 785, 786, 791, 798
Laufendes PSW 91, 280
Laufstatus 93
Laufvariable 481, 583, 584, 586, 596, 597, 703
Laufwerk 334, 350, 351
LBLTYP 338, 339, 652
LBRET 841, 857, 877
LE 355
LEAVE 405, 424, 466, 470, 698, 699
Leeranweisung 403, 411, 426, 427, 451
Leerstelle 419, 560
Leerzeichen 2, 10, 75, 254, 332, 335, 336, 352, 368, 380, 388, 389, 394, 404, 405, 406, 413, 414, 415, 417, 418, 421, 423, 427, 459, 460, 462, 465, 482, 483, 487, 489, 495, 510, 543, 548, 560, 562, 571, 572, 578, 596, 599, 637, 638, 639, 640, 648, 654, 656, 662, 663, 675, 680, 696, 706, 713, 778, 779, 789, 790
Leerzeile 675, 802
LEFT 682, 683
Leiste 679
Leistungsklasse 355
Leitwerk 292
Leitwort 430, 438
LENGTH 532
LERAD 952, 961
Lesebürste 14
Lesedatei 834
Lesefehler 558
Lesen 74, 216, 238, 413, 425, 467, 832, 833, 845, 960
Lesen, binäres 833
Lesen, formatfreies 593, 594
Lesen, selektives 406, 467
Leseoperation 358, 489
Lesestation 5
LESS 651, 711
LIMCT 487
LINE 421, 423, 426, 481
Lineare Gleichung 435
LINES 697
LINESIZE 423
LINK 652, 946
Linkage editor 327, 328, 330, 338, 365, 501, 596, 805, 808, 813, 869, 884, 944
LINKLIB 347, 357, 946, 952

LIST 387, 406, 407, 408, 422, 452, 453
Liste 557
Listenform von Makros 929, 930
Listengesteuert 463, 481, 483
Listengesteuerte EA 404, 405, 406, 407, 414, 415, 530
Listengesteuerte Eingabe/Ausgabe 404, 405, 406, 407, 408, 410, 413, 414
Listenverarbeitung 477, 535
Literal 637, 639, 641, 647, 648, 650, 651, 665, 666, 669, 671, 672, 675, 683, 696, 709, 711, 779, 780, 797, 798, 808, 809
Literal, nichtnumerisches 638, 639, 662, 675, 706, 709, 712
Literal, numerisches 638, 639, 665, 669, 675, 709, 712
Literalangabe 571
Literalpool 809
LNK 329
LNKEDT 336
LOAD 884, 945, 946
Load mode 907
LOCATE 478, 485, 488, 489, 490, 532, 533, 534
Locate mode 902
Lochband 175, 176
Lochbandkanal 826
Locher 1, 3, 4, 7, 388, 768, 769
Locherfunktionen 5, 6
Lochertastatur 3
Lochkarte 1, 2, 3, 13, 14, 15, 56, 113, 327, 332, 335, 349, 351, 352, 380, 386, 406, 407, 419, 447, 468, 471, 474, 475, 495, 542, 543, 545, 559, 634, 635, 636, 652, 694, 697, 700, 701, 773, 800, 802, 825, 931
Lochkartenausgabe 350
Lochkartenformat 335
Lochkartengerät 351, 467, 468
Lochkartenleser 57, 98, 100, 330, 332, 344, 406, 643, 652
Lochkartenmaschine 3
Lochkartenprüfer 14, 15
Lochkartenstanzer 57, 98, 100, 330, 333, 406, 466, 643, 691, 699
Lochkartenstanzgerät 351, 406, 423
Lochkombination 3, 4
Lochschriftübersetzer 14, 15, 47
Lochstation 5, 8
Lochstreifen 9, 386, 545
Lochung 691, 697, 698, 826, 827, 881
LOCK 699, 700
LOG 329, 433
LOG10 433
Logarithmen 37
Logarithmengesetze 37
Logarithmentafel 37, 38
Logarithmus 432, 433, 550
Logarithmus, dekadischer 433
Logarithmus, natürlicher 433
LOGICAL 590
Logik 716
Logisch fortlaufend organisierte Datei 863, 882, 908
Logisch fortlaufend organisierte Datenmenge 343
Logisch fortlaufende Organisationsform 321, 369, 370, 371, 372, 373, 374, 466
Logisch fortlaufende Zugriffsmethode 894
Logische Adresse 267
Logische EA-Einheit 329, 332
Logische Einheit 329, 331, 332, 333, 334, 335, 339, 343, 406, 642, 691
Logische Einheit des Programmierers 329, 331, 878, 887
Logische Einheit des Systems 329, 331
Logische Funktion 44
Logische Instruktion 545, 547, 552
Logische Instruktionen 226-228
Logische Operation 26, 227, 257, 441, 442, 443, 444, 445
Logische Operatoren 26, 226, 227
Logische Programmiereinheit 329, 332, 335
Logische Satzlänge 918
Logische Stufennummer 525, 526, 527
Logische Systemeinheit 336, 344, 345, 863, 984
Logischer Ausdruck 26, 716
Logischer Operator 715, 716
Logischer Satz 305, 308, 309, 823, 826, 827, 833, 909
Logisches IOCS 824, 828, 830, 882, 894
Logisches Zurücksetzen 593
Logout 330
Lokalisierungsmodus 902, 912, 960
Lokalzeit 886
LOW-VALUE 639, 666
LOW-VALUES 639
LSEMCT 811, 812
LSI 51, 329
LT 355
LTORG 809, 813

M-modus 902, 907, 909, 911, 914
MACLIB 347
Magnetband 90, 98, 113, 182, 183, 197, 312, 313, 327, 334, 338, 344, 345, 362, 267, 374, 405, 406, 466, 468, 469, 471, 474, 592, 593, 595, 596, 631, 694, 698, 699, 701, 828, 829, 833, 840, 841, 844, 845, 846, 852, 853, 854, 877, 878, 892, 902, 906, 916, 917, 921, 924, 979, 982, 984, 990
Magnetbandgerät 57, 98, 99, 102, 192, 193, 194, 197, 330, 331, 333, 334, 335, 344, 350, 367, 465, 635, 643, 827, 828, 892
Magnetbandlaufwerk 350
Magnetbandrolle 342, 692, 698
Magnetbandsatz 911
Magnetisierungsrichtung 49, 50, 51
Magnetkern 49, 50, 51
Magnetkernspeicher 49, 52
Magnetplatte 57, 327, 421, 469, 698
Magnetplattengerät 98
Magnetplattenspeicher 344, 374, 465, 467, 592
Magnetplattenspeichergerät 57, 98, 99, 100, 102, 117, 285, 317, 330, 333, 334, 335, 344, 350, 464, 485
Magnetplattenstapel 327, 338, 345
Magnetschrifteleser 102
Magnetstreifenspeicher 593, 694
Magnettrommelspeicher 213, 317, 485
MAIN 387, 402, 403, 404, 505, 507, 538
Makro 339, 777, 819, 823, 825, 826, 828, 829, 836, 838, 841, 842, 849, 853, 854, 857, 859, 860, 863, 864, 866, 868, 869, 872, 875, 877, 878, 884, 885, 886, 888, 892, 900, 903, 904, 905, 906, 907, 908, 909, 919, 926, 930, 937, 938, 939, 940, 944, 948, 949, 950, 951, 956, 957, 958, 960, 961, 962, 976, 977
Makro, Ausführungsform 930
Makro, deklaratives 776
Makro, imperatives 776
Makro, Listenform 929, 930
Makrobefehl 776, 892, 906, 940
Makrobibliothek 347
Makrocode 823
Makrodefinition 976, 977
Makroexpansion 977
Makroinstruktion 823, 883, 887, 892, 893, 921
Makroname 873, 976
Makrosprache 823, 976, 977
Makrostatement 776, 803, 824, 825, 828, 830, 832, 834, 836, 841, 855, 861, 874, 875, 889, 892, 899, 958, 976
Mantisse 25, 28, 216, 218, 219, 220, 222, 244, 392, 638, 664, 795
Marke 399, 401, 403, 418, 419, 430, 440, 447, 448, 449, 451, 495, 498,
Markenbereich 441
Markenkonstante 398, 399, 440, 441
Markenname 399
Markenpräfix 441
Markentyp 482
Markenvariable 439, 440, 441
Markierungslesen, optisches 467, 833
Maschinenadresse 323, 772
Maschinenausrüstung 344, 463
Maschinenbediener 356, 359, 554, 836, 882, 937
Maschinenbedienungspersonal 329, 355
Maschinencode 659, 671, 680, 825, 977, 988
Maschinenfehler 91, 93, 94, 277, 862
Maschineninstruktion 327, 328, 432, 545, 547, 552, 683, 783, 785
Maschinenorientierte Programmiersprache 543, 769
Maschinenorientierte Sprache 543, 769
Maschinenprogramm 543, 769
Maschinensprache 285, 323, 324, 386, 387, 542, 632, 768, 773, 774, 783, 785, 786
Maschinensteuerzeichen 311
Maschinenteil 633, 635, 642, 643, 645, 651, 681, 690, 691, 692, 693, 989
Maske 70, 249
Maskenbit 786
Maskenbitstelle 70, 94
Massachusetts 1
Maßnahme 403, 426, 710
Mathematik 716
Matrix 33, 35, 436, 453, 582, 594
Matrixmultiplikation 436
Matrizenelement 33
Matrizengleichung 35
Matrizenprodukt 34
MAX 433
MEDIUM 405, 406, 407, 409, 465, 467, 468, 481, 489
Medium, externes 415, 416, 417, 427, 435, 463
Mehrfachverzweigung 971
Mehrprogrammbetrieb 334, 344, 595,

836, 849, 887
Mehrprozessorsystem 278, 279, 280
Mehrpunktbetrieb 99
Mehrspuroperation 203, 204, 859
Mehrzeilendrucker 836
MEMORY 690
Merkfeld 318
Merkmalgebiet 345
Merkoperation 367
Messleitung 137
Mikroprogrammierung 284
Mikroprogrammtechnik 284
Minuszeichen 353, 357, 380, 388, 399, 415, 416, 417, 460, 461, 462, 477, 534, 560, 561, 635, 637, 641, 661
Mischprogramm 327
MIT 1
Mitteilungsbyte 861, 862
Mittelwert 30, 31
Mittlere Abweichung 31
Mittlerer Fehler 31
Mnemonischer Name 641, 691, 697, 715
Mnemonischer Operationsschlüssel 62, 237, 245, 274, 298, 323, 769, 774, 823
MODCB 950, 951, 954, 956, 958, 966
MODE 644, 692
Modul 299, 328, 805, 869, 871, 872, 875, 876, 945, 971, 973, 976
Modul komplexer Zahlen 40
Modulende 328
MODULES 690
Modulname 875
Modusattribut 395
Monitoraufruf 245, 282
Monitoreinrichtung 282
Monitorereignis 245, 246, 282
Monitorklasse 282
Monitorschlüssel 282
MOVE 648, 649, 650, 651, 654, 655, 656, 657, 659, 663, 665, 666, 667, 671, 674, 676, 680, 685, 686
Move mode 902
MTMOD 869, 872, 876
MULTIPLE 692, 693, 694, 990
Multiplexkanal 121, 123, 125, 126, 128, 213
Multiplikation 387, 399, 400, 434, 443, 513, 549, 550, 577, 578, 669, 672, 780, 781
Multiplikation einer Matrix mit einem Skalar 33
Multiplikationen von Matrizen 34
Multiplikationsinstruktionen 66, 86, 219, 221, 224
Multiplizieren 399
MULTIPLY 650, 670
Multiprogramming 334, 355, 595
Multiprozessor 92
MVCOM 883

Nachricht 352, 354, 414, 554, 937
Nachsatz 30, 312, 313, 314, 315, 316, 317, 823, 828, 849, 859, 892
Näherungswort 31, 449
NAME 381, 457, 458, 498, 499
NAME-bedingung 427
Name, externer 398, 399, 499, 505, 512, 813, 814
Name, gekennzeichneter 458, 685, 686
Name, Gültigkeitsbereich 396, 498, 499, 509
Name, interner 398, 505, 813
Name, mnemonischer 641, 691, 697, 715
Name, reservierter 433
Name, voll gekennzeichneter 458
NAMED 706
NAMELIST 598, 599
Namensfeld 802, 807, 810, 811
Namensliste 598, 599, 600
Natürliche Zahlen 18, 19
Natürlicher Logarithmus 38, 433
NCP 467
NE 355
NEGATIVE 714
Neudefinieren, einfaches 519
Neudefinition 519, 520, 521, 676, 677, 678, 679, 683, 686
Neudefinition, einfache 521
Neudefinition, korrespondierende 521
Neudefinition, überlagernde 520
Neue Seite 802
Neunerkomplement 27
Neues PSW 90, 91, 247, 280
NEXT 707, 710
Nicht 441, 442
Nicht ausgerichtete Bitkette 392
Nicht minus 787
Nicht null 787
Nichtalphabetische Klasse 713
Nichtausrichtung 522, 523
Nichtindizierte Hauptstruktur 469
Nichtindizierte Variable 478, 583, 587, 588, 589
Nichtnumerische Klasse 713
Nichtnumerische Übertragung 648, 666
Nichtnumerischer Vergleich 712

Nichtnumerisches Literal 638, 639, 662, 675, 706, 709, 712
Nichtnumerisches Zeichen 427
Nichtreserviertes Wort 648
Nichttemporäre Datenmenge 375
Nichtzeichen 388
Niedrigere Rangordnung 444
NO 429, 692, 698, 699, 700
NOINDEXED 381
NOLABEL 405, 468
Nop 787
Nop-Instruktion 70
NOPURGE 383
Normalausgang 583, 584
Normalisierte Gleitpunktdarstellung 217
NOT 25, 26, 651, 713, 714, 715, 716
NOTAPEMK 405, 468
Notation 397, 398
NOTE 675, 839, 850, 852, 854, 920
Notizoperation 367
NOWRITE 466, 489
NRZ-Verfahren 182
NSTD-reels 692
NULL 477
Null 779
Null, führende 368, 392, 415, 416, 460, 462, 487, 562, 578, 664
Nullenschreibung 6
Nullenunterdrückung 460, 461, 662, 663
Nullkette 391, 977
Nullwert 477
NUMERIC 713
Numerische Arbeitsweise 4
Numerische Daten 427, 481
Numerische Klasse 713
Numerische Konstante 793
Numerische Übertragung 648, 666
Numerische Variable 521
Numerischer Teil 46, 80
Numerischer Vergleich 712
Numerisches Datenelement 646, 661, 662, 664, 687
Numerisches Datenfeld 460, 461
Numerisches Feld 458, 460, 571
Numerisches Festpunktdatenelement 707
Numerisches Literal 638, 639, 665, 669, 675, 709, 712
Numerisches Zeichen 2, 46, 637
Numerus 38

Obere Grenze 437, 533
OBJECT-COMPUTER 690
Objektmaschine 632
Objektmodul 328, 329, 330, 810, 813, 869
Objektmoduln, Bibliothek 869
Objektprogramm 386, 400, 403, 404, 410, 414, 435, 453, 505, 509, 528, 538, 542, 543, 546, 547, 551, 552, 553, 554, 555, 562, 574, 581, 584, 588, 590, 591, 595, 596, 632, 633, 698, 709, 769, 770, 776, 777, 783, 790, 798, 801, 803, 805, 895
OCCURS 678, 679, 680, 681, 991
Oder 25, 26, 227, 235, 248, 257, 272, 441, 442
Oderzeichen 388
OF 685, 686
Offene Funktion 551
Offset 328, 527
OFFSET 523
OFL 431
OFLTRACKS 489
Oktalsystem 21, 22
Oktalzahl 21
OMITTED 644, 645
OMR 467
ON 403, 404, 408, 409, 426, 447, 473, 498, 499, 535, 670, 674, 681, 693, 707
On 403, 408
ON-Bedingung 499
ON-Klausel 989
ONCHAR 483
ONCODE 516
Oneinheit 426, 427, 428, 434, 482, 499, 515
ONSOURCE 434, 482, 483
OP 59
OP-Feld 776
OPEN 423, 424, 466, 469, 470, 486, 489, 649, 698, 699, 700, 823, 825, 826, 829, 833, 834, 841, 842, 852, 857, 877, 878, 892, 901, 903, 904, 908, 919, 920, 924, 928, 930, 950
Operand 59, 63, 64, 65, 80, 85, 86, 87, 88, 219, 222, 227, 234, 237, 243, 249, 251, 257, 258, 259, 332, 333, 334, 335, 336, 337, 338, 340, 341, 342, 343, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 364, 365, 399, 409, 441, 442, 444, 669, 672, 674, 707, 716, 774, 777, 778, 781, 784, 785, 786, 789, 794, 797, 803, 805, 806, 809, 814, 826, 827, 834, 840, 841, 842, 849, 851, 857, 860, 869, 870, 874, 875, 876,

878, 883, 885, 886, 888
Operandenfeld 802, 804, 805, 807, 808
Operateur 329, 330, 331, 334, 344, 414, 692, 836, 837
Operateurbefehl 331, 334, 335
Operating system/virtual storage 325
Operation 441, 442, 443, 444, 445, 457, 460, 465, 467
Operation, arithmetische 401, 402, 411, 441, 442, 443, 444, 529, 550, 650, 658, 660
Operation, asynchrone 953
Operation, logische 441, 442, 443, 444, 445
Operation, Rangordnung 444
Operationsschlüssel 56, 59, 62, 91, 108, 109, 233, 237, 249, 323, 769, 785, 786, 787, 807, 873
Operationsschlüssel, erweiterte mnemonische 786, 787
Operationsschlüssel, mnemonischer 323, 769, 774, 823
Operationsschlüssel, symbolischer 769
Operationsschlüsselfeld 776
Operator 25, 26, 355, 399, 400, 409, 440, 444, 549, 651, 669, 672, 711
Operator, arithmetischer 640
Operator, logischer 26, 715, 716
Operatoren, Rangordnung 26
Optimizing compiler 325
Option 339, 402, 407, 408, 413, 414, 415, 421, 422, 423, 424, 426, 452, 457, 458, 469, 470, 478, 479, 480, 481, 484, 485, 486, 487, 489, 490, 507, 512, 513, 522, 527, 532, 533, 534, 535, 536, 558, 681
OPTION 338, 339, 340, 652
OPTION-Steuerkarte 339
OPTION-Steuerstatement 338
OPTIONAL 692
Optionales Wort 640
OPTIONS 387, 402, 403, 404, 505, 507, 538
Optisches Markierungslesen 467, 833
OR 25, 26, 715, 716
ORG 807, 808, 811
Organisationsform 375, 379, 465, 466, 467, 848
Organisationsform von Datenmengen 465, 471, 474
Organisationsform, direkte 368, 369
Organisationsform, gestreute 894
Organisationsform, indexsequentielle 321, 338, 342, 343, 369, 370, 371, 372, 373, 374, 375, 376, 379, 466, 467, 468, 488, 489, 490, 491, 494, 894
Organisationsform, konsekutive 313, 338, 342, 343, 367, 374, 465, 466, 471, 474, 490, 643, 698, 894
Organisationsform, logisch fortlaufende 321, 369, 370, 371, 372, 373, 374, 466
Organisationsform, regionale 338, 342, 369, 466, 468, 485, 486, 491, 494, 922
Organisationsform, starr fortlaufende 313, 367, 465, 466
Organisationsform, wahlfreie 368, 369, 374, 375
Originalmatrix 35
OS 326, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 367, 369, 379, 380, 383, 384, 398, 405, 406, 407, 409, 424, 464, 465, 466, 467, 470, 471, 486, 487, 488, 489, 505, 514, 515, 516, 537, 557, 570, 632, 642, 643, 651, 676, 691, 692, 694, 695, 696, 699, 700, 706, 715, 774, 777, 803, 805, 806, 816, 819, 841, 901, 917, 929, 943, 949, 950, 951, 952, 954, 955, 958, 963, 965, 979, 981, 985, 986, 990
OS/VS 325
OS/VS1 325, 348, 349
OS/VS2 325, 349
OTHERWISE 707, 710
OUTPUT 405, 423, 464, 469, 485, 486, 488, 489, 490, 649, 698
Output class 350
Output writer 349
OVERFL 591
OVERFLOW 427, 428, 431

P-Typ 793, 798
Packen 81, 82
PAGE 347, 348, 421, 423, 481
PAGESIZE 423
PAP 42
Papiertransport 826
Paragraph 636, 642, 674, 675, 690, 691, 695, 697, 703, 706, 708
Paragraphenname 636, 641, 642, 647, 648, 650, 674, 687
Parameter 357, 359, 361, 362, 380, 382, 405, 439, 467, 487, 505, 507,

508, 510, 511, 512, 531, 532, 537, 818, 823, 852, 865, 887, 892, 894, 895, 897, 899, 901, 920, 943
Parameterliste 512, 952
Parität 45, 198
Paritätsfehler 45
Partition 334, 337, 344, 345, 349, 885, 887, 983
Partition, reale 334
Partition, virtuelle 334, 340
Partitionumfang 349
Passwort 376, 949
PAUSE 554, 555
PCH 329
PDUMP 591, 885
PE-Verfahren 182, 184
PER 283, 284
PERFORM 990
Permanente Zuweisung 334
Permanente Datenmenge 348
Pfeil 388, 534
PHASE 340
Phase 339, 340, 345, 884
PHASE-Steuerkarte 340
Phasenkarte 340
Phasenname 340, 884
Phasenverschlüsselung 182, 184
Physikalische Blocklänge 644
Physikalische Satzlänge 925
Physikalisches IOCS 894
Physischer Satz 305, 308
Physisches IOCS 824, 876, 877, 878, 879, 880, 881, 882, 883, 930, 931, 932, 933, 934
PIC 459
PICA 940, 941
PICTURE 459, 480, 481, 521, 523, 645, 646, 648, 654, 655, 657, 659, 663, 665, 671, 677, 678, 683, 684
PIE 940, 941
PL/1 324, 325, 326, 336, 364, 367, 369, 376, 386, 387, 391, 394, 398, 400, 404, 426, 430, 432, 436, 437, 440, 450, 458, 463, 476, 491, 505, 510, 513, 516, 522, 528, 530, 535, 536, 537, 538, 582, 596, 665, 770, 815, 816, 973
PL/1-Programm 387, 389, 402, 465, 470, 515, 537, 986, 987
PL/1-Sprache 433, 449
Platte 700, 823
Plattenbetriebssystem 325
Plattenkennsatz 342, 912
Plattenspeicher 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 274, 367, 592, 593, 596, 694, 829, 849, 853, 854, 865, 867, 877, 882, 892, 923, 924, 979, 990
Plattenspeicherbereich 878, 987
Plattenspeichergerät 197, 209, 210, 211, 212, 213, 214, 691, 828, 829
Plattenspur 848
Plattenstapel 194, 195, 197, 318, 344, 369, 372, 382
Plausibilitätsprüfung 471, 473
PLIDUMP 514, 515
PLILIB 347
PLIOPT 336
PLIRETC 516
Pluszeichen 380, 388, 399, 460
POINT 920, 924, 950, 958
POINTER 476, 477, 523, 532, 534, 535
POINTR 839, 850, 852, 854
POINTS 839, 850, 852, 854
POINTW 850, 852, 854
Polynom 434, 437
Pool 919
POP 803
POSITION 521, 693, 694
POSITIONING 697, 699, 700
Positionsoperand 352, 357, 358, 976
POSITIVE 714
POST 938
Potenz 393, 400, 401, 432, 460, 546
Potenzieren 399
Potenzierung 388, 399, 400, 401, 550, 551, 669
Potenzwert 399, 549, 550
POWER 344, 345
Präfix 429
Präfixfeld 527
Präfixoperator 399
Primärdatenbereich 867
Primärer Eingangspunkt 513
Primärprogramm 542
Primärspur 372
PRINT 383, 384, 405, 421, 464, 591, 802, 803
PRINTER 332, 333
Priorität 355
Privatbibliothek 348
Private Bibliothek 330
Privilegierte Instruktionen 93, 108, 236, 237, 238, 242, 249, 276, 281
PRMOD 869, 871

Problemdaten 390, 391, 392, 393, 394, 410
Problemorientierte Programmiersprache 542, 631, 971
Problemorientierte Sprache 542, 631, 971
Problemprogramm 297, 338, 344, 345, 349
Problemstatus 93, 236, 242
PROC 431, 500
PROCEDURE 387, 402, 403, 404, 426, 429, 431, 500, 505, 507, 513, 531, 537, 538
PROCEDURE DIVISION 633, 647, 648
PROCESSING 692, 693
PROCLIB 347, 363
Produkt 650, 707
Produktzeichen 20
Program interruption control area 940
Program interruption element 940
Programm 42, 56, 69, 90, 91, 95, 290, 301, 323, 324, 325, 326, 327, 329, 332, 335, 336, 337, 338, 339, 340, 342, 343, 344, 345, 347, 348, 349, 355, 357, 358, 359, 362, 363, 364, 365, 367, 374, 379, 381, 384, 386, 387, 388, 389, 390, 396, 397, 398, 400, 402, 403, 404, 408, 409, 413, 414, 426, 428, 430, 431, 432, 434, 437, 439, 440, 441, 447, 448, 449, 453, 465, 466, 469, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 491, 492, 493, 494, 495, 498, 499, 500, 501, 502, 503, 505, 506, 508, 509, 510, 511, 512, 513, 514, 515, 516, 522, 536, 537, 542, 543, 545, 551, 552, 554, 555, 563, 565, 581, 583, 591, 593, 595, 596, 598, 631, 632, 633, 634, 635, 641, 642, 644, 645, 648, 650, 651, 652, 656, 660, 665, 674, 679, 680, 681, 685, 686, 692, 697, 701, 702, 703, 704, 705, 706, 707, 708, 709, 768, 769, 770, 777, 780, 781, 783, 790, 799, 802, 809, 813, 814, 823, 824, 825, 828, 830, 845, 846, 869, 875, 882, 883, 887, 888, 904, 911, 918, 931, 932, 933, 934, 937, 970, 971, 973, 975, 988
Programm-Marke 917
Programm-Maske 94, 230
Programm-Unterbrechungselement 940
Programm, aufrufendes 507, 510, 513
Programm, ausführbares 330, 336
Programm, symbolisches 770
Programm, verschiebliches 869
Programmablauf 419, 423, 450, 485, 515, 531, 554, 575, 706
Programmablaufplan 42, 43, 291, 294, 295, 296, 297, 298, 299, 386, 563, 564, 651, 970, 973
Programmablaufsteuerung 90, 131, 327, 328, 348, 349, 402, 403, 404, 406, 426, 450, 499, 511, 553, 554, 581, 583, 589, 636, 650, 708, 710, 805, 816, 825, 828, 845, 860, 868, 884, 886, 940, 943, 944, 945, 946, 952, 960, 978, 988
Programmablaufzeit 543, 690, 869, 971
Programmabschnitt 328, 406, 409, 418, 449, 450, 495, 501, 533, 585, 797, 810, 811, 812
Programmausführung 336, 337, 364, 402, 431, 453, 485, 581, 636, 676, 694, 876, 973
Programmausrüstung 463
Programmaustausch 570
Programmbedingung 430
Programmbeginn 429
Programmbibliothek 823
Programmbibliothek, verschiebliche 824
Programmeinheit 3, 7, 299, 339, 340, 796, 805, 810, 811, 815, 816, 818, 819, 944
Programmeinheit, aufgerufene 817, 818, 819, 944
Programmeinheit, aufrufende 816, 817, 818, 819, 943, 946
Programmende 428, 509, 824, 851
Programmereignis 245, 283, 284
Programmfehler 885, 955
Programmidentifikation 696
Programmieraufwand 970
Programmiereinheit, logische 329, 332, 335
Programmieren 290, 300, 312, 768
Programmierer 323, 324, 328, 329, 331, 345, 346, 353, 355, 368, 386, 394, 398, 400, 407, 414, 423, 428, 429, 430, 432, 436, 438, 453, 463, 469, 474, 500, 519, 522, 527, 528, 532, 536, 537, 543, 551, 562, 584, 591, 597, 633, 635, 640, 641, 642, 652, 654, 655, 674, 676, 678, 685, 697, 769, 781, 783, 784, 786, 797, 799, 812, 816, 819, 823, 824, 829, 830, 852, 875, 883, 887, 892, 894, 898, 899, 900, 927, 931, 936, 937, 944, 963, 970, 971, 977

Programmiererteam 970, 971, 973
Programmiersprache 323, 324, 325, 326, 332, 376, 543, 632
Programmiersprache / 1 386
Programmiersprache, maschinenorientiert 543
Programmierteam 970
Programmiertechnik 797
Programmierung 327, 369, 390, 510, 543, 631, 770, 823, 844
Programmierung, automatische 769
Programmierung, rekursive 973
Programmierung, strukturierte 971, 973
Programmierungstechnik 477
Programming language/one 324, 386
Programmkarte 5, 7, 8, 9, 11, 14
Programmladen, einleitendes 114, 326, 331, 334, 335, 349
Programmliste 590, 802
Programmname 336, 337
Programmschema 543, 544, 549, 634, 635, 873
Programmsprache 515
Programmstatuswort 90, 91, 93, 110, 131, 230, 235, 238, 280, 886, 934, 940
Programmstelle 426, 429, 430
Programmsteuerabschnitt 328
Programmsteuerhebel 4, 8
Programmsteuerkarte (Lochen) 4, 5
Programmsteuerungsdaten 398, 399, 534
Programmteil 339, 431, 439, 512, 513, 519, 674, 708, 776, 824
Programmtrommel 4, 5, 7, 9
Programmtyp 326
Programmunterbrechungen 90, 91, 99, 105, 237, 241, 245, 246, 268, 429, 940, 941, 971
Programmverbindung 888, 889, 943, 944, 945, 946
Programmverbindung, dynamische 945, 946
Programmverbindung, einfache 943, 944
Projektbericht 970
Protokollierung 951
Prototyp 976
Prozedur 341, 363, 364, 365, 398, 399, 429, 448, 504, 505, 507, 508, 509, 510, 512, 513, 531, 536, 538, 986, 987
Prozedur, aufrufende 507, 508, 510, 512
Prozedur, externe 500, 509, 510, 512
Prozedur, katalogisierte 354, 357, 363, 364, 365, 538
Prozedur, rekursive 509, 510
Prozedurausführung 531
Prozedurbibliothek 341, 347, 363
Prozedurblock 500
Prozedurname 341, 364, 399, 404
Prozedurschritt 364
Prozedurstep 364, 365
Prozedurstepname 365
Prozedurteil 633, 634, 647, 648, 649, 650, 651, 655, 657, 659, 663, 674, 675, 684, 686, 687, 703
Prozedurvereinbarung 508, 512, 513
Prozentzeichen 388
PRTOV 826, 828, 829, 835, 836, 855, 906, 907, 920
Prüfanzeige 592
Prüfanzeiger für Benutzerprogramm 345
Prüfbit 45
Prüfbyte 112, 113, 114, 182, 209, 210, 211, 212, 917, 931
Prüfdaten 198, 199, 200
Pseudoabschnitt 812, 813, 814, 844, 845, 846, 847, 848, 913, 914
Pseudocode 823, 892
Pseudooperationsschlüssel 776
Pseudosatz 922
Pseudovariable 481, 482, 483, 532
PSW 90, 91, 93, 95, 110, 111, 114, 131, 235, 238, 247, 248, 280, 284, 886, 934, 940
PSW, altes 885
PSW, neues 90, 91, 247, 280
Puffer, 103, 381, 465, 478, 479, 484, 515, 527, 533, 693, 694, 825, 899, 901, 918, 919, 921, 924, 925, 928
Pufferfreigabe 928
Puffergebiet 484, 844, 845
Pufferlänge 909
Pufferpool 900, 900, 918, 919
Pufferspeicher 103
Pufferung 464, 490
Pufferung, dynamische 919, 925, 926, 928
Pufferungsattribut 464, 485
Puffervorrat 899
PUNCH 803,
Punkt 356, 637, 680, 691, 779
PURGE 383
PUSH 803
PUT 387, 405, 407, 408, 413, 414, 415, 416, 418, 419, 421, 422, 423, 426, 452, 479, 480, 481, 852, 865, 893, 894, 904, 908, 910, 913, 954, 958, 962

PUTR 836, 837
PUTX 904, 908
Pythagoras 38

QSAM 893, 896, 898, 900, 901, 902, 920
Quadratische Gleichung 447, 448
Quadratische Matrix 33
Quadratwurzel 432, 433
Quellencomputer 690
Quellendatei 405
Quelleneingabestrom 365
Quellenmaschinen 632
Quellenmodul 328, 330, 335, 810, 970, 973
Quellenprogramm 339, 341, 355, 365, 386, 387, 390, 536, 537, 542, 543, 544, 547, 548, 553, 555, 581, 584, 590, 632, 640, 642, 648, 652, 769, 778, 803, 804, 805, 900
Quellenprogrammbibliothek 537
Quellenstatement 330
QUISAM 893, 896, 898, 900, 907, 908
QUOTE 639
QUOTES 639
Quotient 528, 550, 650, 780

R-Formatelement 418, 419
Radiant 39
Randbegrenzung 635, 641, 642, 644, 647, 648, 674, 675
RANGE 396, 397
Rangordnung, höhere 444
Rangordnung 399, 400, 401, 444
Rangordnung der Operatoren 26
Rangordnung für die Operation 444
Rangordnung, niedrigere 444
RCE 405, 467
RDF 377
RDR 329
READ 469, 470, 478, 484, 485, 486, 488, 490, 522, 527, 534, 557, 558, 570, 572, 576, 583, 591, 592, 593, 594, 595, 597, 598, 649, 699, 852, 853, 854, 855, 857, 858, 859, 860, 861, 866, 893, 894, 897, 918, 919, 920, 921, 922, 924, 925, 926, 927, 928, 929, 930
READER 332, 333
READY 706
REAL 395, 396, 397, 398, 401, 409, 433, 482, 506, 507, 548, 550, 582, 584, 589, 590
Reale Adresse 267, 269, 279
Reale Partition 334
Realer Speicher 337
Realer Speicherraum 979
Realmodus 888
Realspeicher 267, 270, 323, 327, 337, 344, 349, 353, 354, 374
Realteil 40, 41, 395, 418, 459, 482
REC 329, 863
Rechenbedingung 427
Rechenoperation 660, 669
Rechensprunganweisung 587, 588
Rechenwerk 57
Rechner 44
Rechteckige Matrix 33, 50
Rechtsbündigkeit 795
Record 368
RECORD 464, 465, 644, 649, 693, 694, 699
Record Definition Field 377
Record-Overflow 693, 694
RECORDING 644
RECORDS 644
RECORDSIZE 381, 382
RECSIZE 405, 465
RECURSIVE 509
REDEFINES 676, 677, 678, 680, 684, 686
REEL 692, 699, 700, 990
Reelle Daten 392
Reelle Konstante 546, 547
Reelle Variable 547, 548, 560, 576, 579, 598
Reelle Zahlen 18
Reentrant 973
REFER 532, 533
Referenztabelle 966
Region 198, 487
REGIONAL 465, 466, 467, 486, 487
Regional organisierte Datei 857
Regionale Organisationsform 338, 342, 369, 466, 468, 485, 486, 491, 494, 922
Register 55, 59, 65, 73, 89, 95, 222, 224, 225, 227, 232, 233, 235, 236, 237, 251, 258, 279, 675, 676, 781, 782, 783, 797, 815, 819, 829, 899, 977
Register, allgemeines 781, 826
Register, spezielles 675, 676, 692, 696, 991, 992
Registername 675
Registernotierung 852, 868, 876, 878, 883, 887, 901, 904, 905, 906, 907,

919, 921, 928, 929, 935, 936, 937, 938, 939, 944, 958
Registernummer 779
Registerreferenz 966
Regressionskoeffizient 32
Reihe 13
Reihenfolgefehler 863, 867, 882
Reihennummer 360, 379, 383
Reihenweise EA 415, 427, 452, 464, 475, 481, 530
Rekursiv 509, 973, 974, 975
Rekursive Funktionsprozedur 509, 510
Rekursive Programmierung 973
Rekursive Prozedur 509, 510
Relative Adresse 60, 781
Relative Byteadresse 377, 958, 959
Relative Satzadresse 927
Relative Satznummer 486, 487
Relative Spuradresse 927
Relative Spurnummer 368, 932
RELEASE 887
Relocation Dictionary 328
RELSE 828, 904, 905, 908
REMAINDER 670
RENAMES 684
Reorganisation 373, 375
Reorganisationskriterium 908
Report Program Generator 324
REPRO 802, 803
REPY 414
REREAD 405, 424, 466, 470, 698, 699
RES 329
RESERVE 692, 693
Reservierter Name 433
Reserviertes Wort 389, 394, 638, 674, 675, 710
RESET 706
Restart 92, 94
Restartunterbrechung 92
Resultatfeld 671
Rettungsgebiet 815, 816, 886, 889, 935, 937, 942, 943, 964, 974, 975
RETURN 404, 426, 428, 499, 511, 512, 888, 889, 911, 943, 945
RETURN-CODE 676
RETURNS 506, 507, 508, 512, 513
REVERSED 698
REWIND 698, 699, 700
REWRITE 484, 485, 486, 488, 490
RIGHT 682, 683
RLB 329
RLD 328
Rootphase 340
ROUNDED 669, 670, 672
Routine 298, 978
RPG 324, 325, 596
RPL 950, 953, 954, 956, 961
RR-Format 59, 63, 64, 65, 66, 71, 72, 73, 94, 217, 218, 227, 230, 235, 291, 783
RS-Format 224, 225, 233, 261, 784
Rückkehr 426, 427, 483, 510
Rückkehradresse 781, 815, 907, 946, 952
Rückkehrcode 515, 676, 817, 943, 948, 952, 955, 958, 961, 986
Rückkehrpunkt 975
Rückkehrstelle 818, 975
Rücksetztaste 7
Rücksprung 816, 943, 944
Rücksprungadresse 842
Rückspulen 335
Rückstellstapel 536
Rückwärtslesen 465, 839, 904, 917, 921
Rückwärtszeiger 535
RUN 636, 650, 676, 709
Rundungsfehler 28, 29
RX-Format 60, 63, 64, 65, 66, 69, 71, 72, 73, 94, 217, 218, 224, 225, 227, 230, 261, 264, 783

S-Format 108, 125, 127, 129, 261
S-Typ 796, 797
SAME 693, 694
Satz 15, 305, 308, 309, 327, 360, 361, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 422, 423, 463, 465, 466, 467, 468, 469, 471, 473, 478, 484, 486, 487, 488, 489, 494, 495, 527, 535, 576, 593, 594, 599, 635, 636, 644, 645, 647, 649, 650, 651, 674, 675, 676, 683, 694, 695, 696, 697, 708, 826, 828, 849, 857, 860, 861, 862, 863, 866, 867, 868, 909, 924, 925, 926
Satz fester Länge 645, 903
Satz unbestimmter Länge 903
Satz undefinierter Länge 645, 650, 833
Satz variabler Länge 645, 650, 903
Satz, geblockter 367, 374, 375, 467, 527, 838, 841, 842, 850, 851
Satz, logischer 305, 308, 309, 823, 826, 827, 833, 909
Satz, physischer 305, 308
Satz, ungeblockter 369, 374, 375, 486, 487, 489, 830, 866, 868, 870
Satzadresse 927

Satzadresse, relative 927
Satzart 307, 308, 309, 310, 311, 367, 368, 369, 405, 644, 647, 650, 676, 681, 832, 835, 837, 839, 927
Satzbereich 198, 199, 317, 369, 371, 373, 595
Satzbeschreibung 644, 989
Satzblock 828
Satzdefinitionsfeld 377
Satzeinheit 15
Satzeinheitgerät 643, 645
Satzform 307, 308, 309, 310, 311, 832, 835, 837, 839
Satzformat 307, 308, 309, 310, 311, 465, 487, 489, 839, 865
Satzgebiet 676
Satzkontrollfeld 308, 309
Satzlänge 305, 307, 311, 375, 381, 405, 527, 596, 644, 681, 694, 839, 841, 851, 852, 853, 855, 865, 866, 897, 898, 918, 948, 956, 957, 962, 979
Satzlänge, falsche 105, 106, 114, 117, 119, 202, 829, 861, 862, 867, 925
Satzlänge, feste 307, 310
Satzlänge, logische 918
Satzlänge, unbestimmte 307
Satzlänge, undefinierte 307, 918
Satzlänge, variable 307, 310
Satzlänge, physikalische 925
Satznummer 368, 369, 495, 594, 854
Satznummer, relative 486, 487
Satzschlüssel 374, 909, 925, 954, 963
Satzweise EA 464, 467, 471, 472, 473, 477, 478, 479, 522
Satzweise Eingabe 469
Satzweise Eingabe/Ausgabe 463, 464, 465, 466, 467, 468, 469, 470
SAVE 829, 888, 889, 911, 943, 945
Save Area 815
SCALARVARYING 467
Scan Mode 907
Schablone 253
Schalter 440, 715
Schalterzustand 715
Schaltkreis 51, 52, 284, 285
Scheinelement 527
Scheingerät 351
Scheinsatz 487, 922
Scheinvariable 976
Schiften 88, 233, 234
Schleife 232, 297, 408, 444, 449, 450, 451, 452, 583, 584, 585, 636
Schleife mit anormalem Ausgang 971, 972
Schleife, induktive 583, 584, 585
Schleife, iterative 450
Schleifendurchlauf 583
Schleifennotation 596, 597
Schleifenspanne 583, 584, 585, 586
Schließen 955, 982
Schlüssel 199, 206, 368, 369, 370, 371, 372, 373, 374, 375, 376, 378, 379, 381, 465, 486, 487, 488, 489, 853, 857, 859, 860, 861, 865, 866, 867, 868, 882, 896, 909, 926, 927, 928, 954, 955, 959, 960, 962, 963
Schlüssel, aufgezeichneter, 487
Schlüssel, eingebetteter 376, 467, 489
Schlüssel, generischer 466, 488, 909, 954
Schlüsseldatenmenge 378
Schlüsselfeld 199, 200, 306, 318, 369, 370, 371, 372, 374, 381, 861, 862, 928, 948, 963
Schlüsselfolge 375, 377, 378, 379, 381, 382, 490, 954, 957, 958, 959, 960, 963
Schlüsselkomponente 382
Schlüssellänge 466, 467, 866, 908, 909, 962
Schlüsseloperand 342
Schlüsselposition 488
Schlüsselsatz 379, 381
Schlüsselvariable 486
Schlüsselvergleich 376
Schlüsselwertbereich 962
Schlüsselwort 335, 360, 361, 381, 382, 398, 403, 411, 412,430, 438, 451, 452, 453, 469, 500, 596, 640, 641, 669, 671, 674, 685, 698, 707, 711, 957, 958
Schlüsselwortoperand 352, 353, 953, 956, 976
Schlußabschnitt 297
Schnittstellerroutine (VSAM-ISAM) 376
Schrägstrich 334, 335, 352, 388, 538, 573, 574
Schreib/Lese-Kopf 182, 212, 320, 343, 368, 592
Schreiben 201, 238, 413, 425, 467, 925, 926
Schreiben, formatfreies 593, 594
Schreiber 350
Schreibmaschine 3, 43, 44, 47, 98, 174
Schreibprüfung 694, 851, 859, 865, 957
Schreibprüfungsfehler 881
Schreibring 184, 185
Schreibweise 397, 398
Schwaches externes Symbol 814

Scientific Sabroutine Package 505
SD 342
SDMOD 869, 872, 873
Secans 39
SECTION 644, 648, 690, 692
Sedezimale Adresse 797
Sedezimale Konstante 791, 792, 793
Sedezimaler Direktwert 779
Sedezimalkonstante 792
Sedezimalsystem 21, 22
Sedezimalzahl 21, 22, 25, 216, 340
Sedezimalziffer 22, 46, 59, 61, 81, 107, 217, 218, 227, 340, 658, 777, 779, 793, 803, 983
SEDZ 107, 212, 213, 214, 326, 341, 342, 348, 360, 363, 367, 368, 369, 374, 375, 378, 379, 382, 468, 485, 486, 488, 596, 699, 827, 848, 849, 878, 879, 880, 881, 892, 902, 916, 917, 920, 927
Segment 54, 267, 268, 269, 270, 305, 309, 310, 971
SEGMENT-LIMIT 690
Segmentfolgenummer 268
Segmentgrenze 690
Segmentgröße 267, 268
Segmentnummer 268, 270, 272
Segmenttabelle 244, 267, 268, 272, 273
Segmentumsetzung 246, 268, 269, 270
Seite 266, 271, 421, 423, 454
Seite, neue 802
Seitenbibliothek 347
Seitendatei 349
Seitenende 905
Seitenfehlbedingung 267
Seitenfehler 267
Seitenfolgenummer 268
Seitengrenze 936
Seitengröße 267, 268
Seitennummer 268, 269, 270, 272
Seitensupervisor 271
Seitentabelle 269, 271, 272, 273
Seitenumfang 937
Seitenumsetzung 244, 246, 269, 270
Seitenwechsel 271, 327, 344, 349, 351, 979
Sekretär 970, 971
Sekundäre Eingangsstelle 514
Sekundäre Zuordnung 380
Sekundärer Eingangspunkt 513
Sekundärprogramm 542
Sekundärzylinder 380
Selbst definierendes Glied 779
SELECT 642, 692
Selektisches Lesen 406
Selektives Lesen 467
Selektorkanal 99, 102, 124, 185
Semikolon 387, 388, 403, 411, 430, 674
Sendefeld 654, 656, 658, 663, 664, 665, 666, 676, 678, 991
Seniorprogrammierer 977
SENTENCE 707, 710
SEQUENTIAL 464, 465, 490, 692
Sequentiell 963
Sequentielle Verarbeitung 825, 826, 827, 828, 829, 830, 868, 869
Sequentielle Zugriffsmethode 367, 375, 464, 478, 479, 484, 485, 486, 488, 489, 490, 894
Sequentieller Zugriff 367, 595, 596, 864, 954
Sequentielles Laden 864
Seriell wiederverwendbar 973
Seriennummer 360
SET 478, 485, 490, 522, 527, 533, 534, 536
SETEL 866
SETIME 86
SETL 869, 909, 910
SHOWCB 951, 956, 597, 958, 966
SI-Format 225, 226, 227, 261, 282, 784, 785
SIGN 690
SIGNAL 429
Signifikante Ziffer 578
Signifikanzfehler 95
Simulator 941, 942
Simultaneous Peripheral Operations On Line 344
SIN 433
SIND 433
Sinus 39, 433
SIO 917
SIZE 428, 429, 670, 671, 673, 674, 690
Skalar 18, 33
Skalare Ergibtanweisung 436, 457
Skalare Variable 451, 455, 457, 482, 509
Skalenfaktor 416, 460, 577, 578, 579, 794, 795, 798
Skalierungsattribut 395, 399
SKIP 421, 422, 423, 426, 481
SLB 329
SLITE 592
SLITET 592
SNAP 426, 935
Sofortvorschub 572, 827

Software 463, 465, 570
Sonderzeichen 2, 46, 47, 353, 357, 388, 389, 547, 548, 637, 640, 655, 778, 779
Sortierausgang 988
Sortierbegriff 980
Sortierblock 979
Sortieren 978, 979, 984, 986, 988
Sortierfolge 409
Sortierlauf 979, 983, 987
Sortiermaschinen 1, 15, 16
Sortierprogramm 327, 978, 979, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990
Sortierprozedur 992
Sortierstatement 980, 981
Sortierverfahren 992
Sortierzeit 988
SOUCRE-COMPUTER 690
SPACE 379, 380, 638, 639, 802
SPACES 639, 666, 667
Spalte 776, 804
Spalten (von Lochkarten) 2, 5, 14, 103
Spaltenanzeiger 7, 11
Spaltenunterdrückung 833
Spaltenvektor 33
Spannsatz 306, 309, 310, 645, 903, 904, 922
SPECIAL-NAMES 690, 691, 695, 697, 715
Speicher 344, 887
Speicher, reale 337
Speicher, realer 270
Speicher, virtueller 331, 337, 344, 349, 351, 353, 354, 374, 375, 888, 982
Speicherabzug 934
Speicherauszug 591, 934
Speicherauszugsform 383
Speicherbedarf 543, 549
Speichereinheit mit direkter Zugriffsmöglichkeit 326, 348, 360, 368, 468, 485, 699, 849
Speichereinteilung 334
Speichergebiet 888
Speichergerät mit direkter Zugriffsmöglichkeit 351, 464, 465, 828
Speichergerät, externes 351, 633, 643, 645, 824
Speicherkapazität 54
Speicherklasse 501, 505, 535
Speicherklasse, automatische 505, 531, 535, 973
Speicherklasse, basisbezogene 535
Speicherklasse, gesteuerte 535, 536
Speicherklasse, statische 535, 973
Speicherklassenattribut 500, 503, 504
Speichermakro 887, 888
Speichermedium, externes 419, 420
Speichern 43, 44, 57
Speicherplatz 323, 324, 329, 337, 374, 375, 379, 381, 431, 435, 500, 503, 510, 519, 520, 532, 533, 534, 535, 536, 550, 551, 552, 581, 588, 770, 776, 777, 790, 797, 798, 805, 808, 810, 834, 848, 883, 948, 985
Speicherplatz, virtueller 339
Speicherraum, 379, 380, 382, 383, 887, 888
Speicherschlüssel 238, 239
Speicherschutzschlüssel 93, 107, 238, 242, 280
Speichertechnik, virtuelle 325
Speicherumfang, virtueller 337
Speicherung 441
Speicherung, temporäre 351
Speicherungsinstruktionen 65, 220, 225, 235, 237, 250, 279
Speicherwerk 57
Speicherzyklus 52
Sperrfrist 341, 342
Spezielles Register 675, 676, 692, 696, 991, 992
Spezifikation 452, 453
Spezifikationsfehler 242
Spezifikationsliste 453
Spezifikationsstatement 593
Spezifikationsumsetzung 245
Spezifikationsvereinbarung 545, 547, 552
SPIE 940, 941
Spool 344
Spoolbereich 344
Spooldatei 349
Spooldatenmenge 344, 347, 348
Sprache, absolute 768
Sprache, maschinenorientiert 769
Sprache, problemorientiert 542
Sprachebene 542
Sprachelement 422, 484, 535, 570
Sprachübersetzer 326, 327, 328
Sprung 554, 588, 971
Sprung, gesetzter 589
Sprung, unbedingter 708
Sprunganweisung 499
Sprunganweisung, unbedingte 426
Sprungziel 708
Spulenetikett 312
Spur 195, 197, 198, 320, 321, 343, 360, 368, 369, 370, 371, 372, 373, 374,

378, 379, 380, 468, 486, 487, 854, 857, 859, 861, 862, 897, 922, 928
Spuradresse 197, 198, 200, 318, 927
Spuradresse, relative 927
Spurbeschreibungssatz 197, 200, 320, 922, 927, 929
Spurende 372
Spureninhaltsverzeichnis 371, 373
Spurenüberlauf 199, 320, 880, 892
Spurenüberlaufbereich 373
Spurenüberlaufeinrichtung 467, 694, 903, 926
Spurenüberlaufsteuersatz 372
Spurenverbindung 199, 694, 892, 926
Spurindex 371, 375, 379
Spurkapazität 866
Spurnummer 368, 923
Spurnummer, relative 368, 932
SQRT 432, 551
SS-Format 79, 81, 85, 88, 94, 227, 261, 785
SSP 505
STACKER 405, 468
STANDARD 644, 645
Standardabweichung 30, 31, 409
Standardattribut 396, 397, 464, 507
Standardausgabedatei 421
Standardausgabeschreiber 359
Standarddatei 407, 408
Standarddateiname 408
Standarddruckdatei 408, 409, 415, 416, 422, 426, 428
Standardeingabegerät 348
Standardetikett 335, 338, 341, 367
Standardgenauigkeit 443
Standardgenauigkeitsattribut 395
Standardisierte IBM-Prozedur 365
Standardkennsatz 335, 338, 341, 362, 367, 596, 694, 849, 852, 877, 878, 892, 982, 983, 990
Standardlänge 795
Standardmaßnahme 426
Standardoption 338, 339
Standardsteuerstatement 363
Standardsystemausgabegerät 330
Standardsystemeingabegerät 330
Standardsystemmaßnahme 426, 430, 473, 514
Stanzdatei 834
Stanzen 56, 74, 103, 349, 467, 697, 698, 804, 830, 833, 897
Stanzen in gleiche Karte 351
Stanzen in gleicher Karte 147
Stanzer 350, 832
Stanzfehler 831
Starr fortlaufend organisierte Datei 595, 596, 857, 919
Starr fortlaufende Organisationsform 313, 367, 465, 466
START 349, 811, 812, 848
Startbefehl 349
Startpunkt 958
Statement 324, 332, 333, 334, 335, 338, 341, 344, 345, 356, 357, 358, 364, 365, 380, 382, 383, 384, 386, 387, 394, 396, 397, 398, 402, 403, 404, 405, 406, 407, 409, 411, 412, 413, 417, 418, 419, 421, 422, 423, 426, 429, 430, 435, 436, 437, 440, 441, 444, 447, 448, 449, 450, 451, 452, 453, 455, 458, 466, 467, 469, 478, 480, 481, 483, 484, 485, 486, 487, 488, 489, 490, 495, 498, 505, 509, 511, 512, 513, 514, 515, 533, 534, 535, 536, 537, 543, 545, 554, 555, 557, 570, 571, 573, 577, 581, 583, 584, 585, 587, 588, 589, 591, 592, 595, 596, 598, 600, 636, 648, 649, 651, 656, 659, 663, 665, 667, 669, 671, 674, 675, 676, 684, 685, 695, 697, 698, 699, 706, 707, 708, 710, 711, 714, 715, 774, 776, 777, 782, 785, 789, 790, 794, 796, 797, 799, 803, 804, 806, 809, 813, 825, 853, 869, 875, 892, 971, 973, 979
Statement, arithmetisches 669, 670, 671, 678
Statement, einfaches 426
Statement, verschiebliches 330
Statementart 776
Statementfolge 356, 450, 452, 495, 499, 500, 529, 650, 667, 686, 707, 710, 976
Statementform 490
Statementgruppe 447, 448, 449, 452
Statementnummer 543, 554, 555, 558, 559, 583, 584, 587, 589
STATIC 500, 501, 503, 505, 535
Statische Kürze 290
Statische Speicherklasse 535, 973
Statisches Abbildungszeichen 462
Statistik 30
Status, unwirksamer 429
Status, wirksamer 428, 429, 528
Statusbyte 107, 108, 109, 112, 113, 123, 124, 125, 127, 132
Stellenverschiebung 89, 233, 234
Step 355, 356, 357, 364, 365, 384

STEPCAT 358, 384
Stepende 650
STEPLIB 358
Stepname 356, 357, 652
Sternzeichen 340, 361, 388, 399, 400, 438, 460, 462, 560, 562, 637, 664, 675, 781, 782, 806
Steuerabschnitt 328
Steueranweisung 545, 553, 554, 555, 582, 583, 802
Steueranweisung für den Assemblierer 777, 778, 801, 802, 803, 804, 805, 806, 807
Steuerblock 515, 951, 953, 957
Steuerbyte 488, 644
Steuereinheit 98, 100, 102, 123, 125, 126, 128, 137, 198, 202, 207
Steuergebiet für Programmunterbrechungen 940
Steuergerät 98
Steuerinformation 377
Steuerkarte 327, 335, 338, 339, 340, 356, 406, 850, 852, 869, 878, 979, 980, 981, 983, 984, 985, 986
Steuerkonsole 286, 287
Steuerprogramm 326, 327, 341, 681, 816
Steuerregister 55, 93, 95, 110, 111, 237, 245, 265, 267, 268, 270, 282, 283
Steuersatz 327, 330, 335, 336, 341
Steuersequenz 971
Steuerstatement 332, 334, 335, 338, 339, 341, 347, 348, 349, 356, 358, 363, 380, 383, 386, 463, 466, 467, 470, 514, 537, 652, 691, 862, 988
Steuerung 351
Steuerungsformatelement 415, 419, 421, 422, 454
Steuerungsoperation 43, 57, 201
Steuerungssatz 693
Steuerungsstatement 404
Steuerwerk 57
Steuerzeichen 311, 405, 406, 422, 571, 575, 576, 697, 698, 703, 833, 836, 855
STIMER 939
STOP 404, 428, 499, 515, 555, 591, 636, 650, 676, 709
STREAM 464
Streuung 30, 31
STRING 479, 480, 481
STRINGRANGE 429
STRINGSIZE 429
Struktur 455, 456, 457, 458, 466, 474, 475, 476, 478, 500, 504, 519, 521, 522, 523, 524, 525, 526, 527, 532, 533, 534, 535, 536, 971, 972
Struktur, basisbezogene 477, 532, 534, 535
Struktur, einfache 972
Strukturelement 456, 457, 458, 476, 500, 504, 505, 526, 527, 532, 533, 534
Strukturergibtanweisung 457
Strukturerklärung 456, 525, 527, 537
Strukturgliederung 457
Strukturierte Programmierung 971, 973
Strukturstufe 456
Strukturteil 524, 526, 527
Strukturvariable 457, 477, 505, 509
Strukturvariable, basisbezogene 533
Strukturvereinbarung 524
Stufe 444
Stufennummer 455, 457, 469, 478, 521, 524, 525, 526, 645, 647, 649, 665, 669, 674, 677, 679, 682, 683, 684, 685, 686, 687, 699, 714, 715
Stufennummer, logische 525, 526, 527
STXIT 885, 886
SUBALLOCATION 380, 381, 382, 383
Subroutine 510, 511, 512, 513, 515, 516
Subroutinename 512
Subroutineprozedur 510, 511
Subroutineprozedur, interne 512
Subroutineunterprogramm 514
SUBSCRIPTRANGE 429
Subset-Kompilierer 632, 690
Subsetsprache 695
Substitute Mode 902
SUBSTR 481, 482
SUBTRACT 650, 669, 670, 671
Subtrahieren 399
Subtraktion 399, 549, 550, 669, 672, 780
Subtraktion von Matrizen 33
Subtraktionsinstruktionen 64, 85, 219, 221, 224, 236
Suchadresse 861, 929, 860
Suchargument 867, 953, 954
Suchen 201
Suchschlüssel 374
Summe 650
Summenindex 20
Summenzeichen 20
Supervisor 92, 94, 235, 326, 327, 329, 331, 338, 344, 349, 351, 636, 845, 883, 884, 885, 886, 887, 936, 940, 943, 945, 946
Supervisoraktivität 973

Supervisoraufruf 235, 236
Supervisormakro 845
Supervisormakroinstruktion 883, 884, 885, 886, 887
Supervisorprogramm 111, 112, 236, 299
Supervisorstatus 93, 94, 108, 236
Supervisorunterbrechung 236
SVC 367, 892
SVC-Bibliothek 347
SVCLIB 347
Symbol 323, 462, 477, 520, 768, 769, 778, 786, 790, 799, 806, 809
Symbol, externes 814
Symbol, schwach externes 814
Symbol, verschiebliches 796
Symbolische Adresse 852
Symbolische Instruktion 776
Symbolischer Einheitenname 331
Symbolischer Operationsschlüssel 769
Symbolisches Programm 770
Symboltabelle 771, 773, 774, 778, 802, 806, 807, 809, 877
Symbolverzeichnis, externes 328, 329
SYNC 683
Synchroner Zugriff 954
SYNCHRONIZED 682, 685
Syntax 388, 389, 390
SYSABEND 358
SYSCAT 863
SYSCHK 358
SYSCLB 863
SYSCP 351
SYSCTLG 347, 348
SYSDA 351
SYSIN 331, 335, 348, 363, 365, 383, 403, 404, 407, 413, 425, 691, 695
SYSIPT 330, 331, 333, 335, 336, 338, 406, 557, 691, 695, 825, 833, 855, 863
SYSJOBQUE 347, 348
SYSLNK 330, 339, 863
SYSLOG 330, 334, 336, 557, 691, 836, 841, 863, 878
SYSLST 330, 331, 335, 344, 406, 514, 557, 691, 696, 706, 855, 863, 885
SYSOUT 331, 334, 335, 349, 359, 691, 696, 706
SYSPCH 330, 331, 333, 335, 344, 406, 557, 691, 696, 833, 855, 863
SYSPOOL 347, 348
SYSPRINT 365, 383, 407, 409, 413, 415, 416, 421, 426, 428
SYSPUNCH 691, 696
SYSRDR 330, 331, 332, 333, 335, 336, 344, 406, 833, 855, 863
SYSREC 330
SYSRES 330, 331, 537, 805
SYSRLB 863
SYSSLB 537, 805, 863
SYSSQ 351
SYSTEM 426
System 330, 331, 351, 356, 361, 362, 427, 435, 455, 516, 887
System Reader 348
Systemausgabeschreiber 350
Systembedingung 428
Systembibliothek 805
Systemdrucker 591
Systemeingabeeinheit 691
Systemeinheit 332
Systemeinheit, logische 336, 344, 345, 863, 984
Systemelement 348
Systementwurf 970
Systemgenerierung 338, 339, 345, 351
Systemkartenleser 591
Systemkartenstanzer 591
Systemkatalog 347, 360, 361
Systemleser 330, 348, 349, 350
Systemmaske 92
Systemmaßnahme 426, 427, 428
Systemname 692
Systemplattenstapel 329
Systemprogramm 348, 349
Systemprogrammierer 341, 348, 351, 352, 355, 357, 360, 384, 515, 557, 558, 937, 985
Systemresidenz 341, 344, 551
Systemschreiber 350
Systemstanzgerät 330
Systemsteuerpunkt 836
Systemtest 970
SYSUDUMP 358
SYSUI1 365
SYSVIO 351
SYSVIS 331, 863
SYS1 347, 348, 357, 359, 363

Tabelle 679, 680
Tafel 679
Tageszeit 676, 939
Tangens 39
TAPE 351, 693
Tape Label 341
TASK 523
Task 973
Tastatur 3, 15, 168, 169, 170, 171

Tastatur (Locher) 3
Tasten 970
TCLOSE 950
Teil 674
Teildatenmenge 378, 379
Teilfeld 460, 461, 462
Teilfolge 463
Teilkette 481
Teilname 458, 642, 674
Temporäre Datenmenge 348, 361
Temporäre Speicherung 351
Temporäre Zuweisung 333
Term 399, 778, 780
Testausrüstung 970
TESTCB 950, 954, 957, 958
Testdaten 970
Testen 706
Testhilfe 514
Testhilfestatement 706, 707
Testphase 970
Text 328
THAN 651, 711
THEN 387, 411, 412, 440, 444, 447, 448, 451, 477, 536, 650, 651, 674, 710
THRU 683, 684, 685
TIME 696, 939
TIME-OF-DAY 676
Tischrechner 23, 27, 44
Titel 802
TITLE 470, 802
TLBL 341, 342, 464, 470, 852
TO 381, 387, 403, 451, 452, 644, 650, 651, 655, 656, 657, 659, 663, 665, 666, 667, 669, 670, 671, 674, 685, 686, 692, 711
TOTAL 466
TRACE 706
TRACK 368
TRACKS 380, 381, 382
Transistor 51
TRANSMIT 427
Transponierte Matrix 34
Trennungszeichen 394
Trennzeichen 415
Trigonometrie 38
Trigonometrische Funktionen 38, 39
TRKOFL 467
Trommelspeicher 213, 593
TRUNC 828, 842, 848, 851, 906
TTIMER 940
TXT 328
Typ, arithmetischer 441, 451, 482
Typfestlegung 590
Typvereinbarung 548, 552, 589
Typvereinbarung, explizite 548, 552, 582, 590

U-Format 903
Uchadresse 860
UCS 827
Überlagernde Neudefinition 520
Überlagerung 339, 340, 521
Überlappung 306, 484, 845
Überlauf 71, 72, 321, 867
Überlaufbereich 343, 369, 372, 373, 374, 867, 908
Überlaufsatz 372, 373, 468, 908, 910, 925
Überlaufspur 321, 370, 489
Überschrift 570, 571, 577, 802
Übersetzen 363, 364
Übersetzer 325, 336, 338, 344, 364, 387
Übersetzername 336, 338
Übersetzerzeit 536
Übersetzung 323, 335, 338, 364, 392, 560, 632, 640, 642, 794
Übersetzungsinstruktionen 251, 252, 253
Überspringen von Zeilen 802
Übertragung 648, 649, 654, 655, 656, 657, 658, 660, 662, 663, 665, 666, 667
Übertragung, nichtnumerische 648, 666
Übertragung, numerische 648, 666
Übertragungsart 666
Übertragungsfehler 862
Übertragungsmodus 902, 904
UFL 431
Uhrzeitgeber 275, 276, 279
Uhrzeitvergleicher 247, 248, 277, 278
Umorganisation 374
Umschaltung 359
Umwandlung 401, 410, 419, 427, 443, 462, 469, 507, 632
Umwandlungsfehler 420
UNALIGNED 522, 523, 524
Unbedingte Sprunganweisung 426
Unbedingte Verzweigung 70, 554, 787
Unbedingter Sprung 708
Unbestimmte Länge 903
UNBUFFERED 464, 484, 485, 490
Und 25, 26, 227, 235, 272, 441, 442, 444
UNDEFINEDFILE 427
Undefinierte Länge 487, 645, 859, 872
Undefinierte Satzlänge 918
UNDERFLOW 428, 431
Undzeichen 388, 789
Ungeblockter Satz 369, 374, 375, 486, 487, 489, 830, 853, 866, 868, 870
Ungepufferte Datei 490

Ungepufferter Zugriff 367
UNIQUE 380, 381
UNIT 692, 699, 700
Universaltypensatz 175, 177, 178, 332, 359, 827, 879
UNLOAD 405, 424, 466, 470
Unnormalisierte Gleitpunktdarstellung 217, 220
UNSPEC 481, 482
Unterbereich 436
Unterbrechung 90, 91, 110, 111, 117, 122, 123, 124, 127, 129, 241, 242, 243, 244, 245, 246, 247, 326, 327, 429, 815, 885, 939, 940
Unterbrechung, externe 90, 91, 238, 276, 885, 886
Unterbrechungsarten 90, 91, 247
Unterbrechungsbedingung 942
Unterbrechungsbehandlung 885
Unterbrechungsklasse 885
Unterbrechungsroutine 940
Unterbrechungsschlüssel 94, 241, 242, 247, 248, 268, 940, 941, 942
Unterbrechungstaste 92, 247, 248, 885
Untere Grenze 437
Untergegliederte Datenmenge 304, 361, 806
Untergeordnete Instruktion 257, 258
Unterkanal 93, 113, 119, 120, 121, 125, 126, 128, 129, 130, 137
Unteroperand 950, 951, 953
Unterparameter 360, 361, 380
Unterprogramm 294, 298, 299, 327, 336, 339, 462, 511, 512, 513, 514, 515, 516, 530, 531, 586, 770, 781, 797, 816, 818, 819, 973, 974, 986
Unterroutine 298, 299
Unterstreichungszeichen 388, 389
Unterstruktur 456, 458, 500, 525
UNTIL 707
Unwahr 409
Unwirksamer Status 429
UPDATE 464, 465, 466, 484, 485, 486, 488, 489, 490
UPON 640, 641, 695
UPSI 715
UPSI-Byte 345, 346, 715
UR 643
USAGE 645, 646, 648, 654, 655, 656, 659, 660, 661, 665, 671, 679, 682
USASCII 48
USASI 405, 422, 542
User Program Sense Indicator 345, 347, 348, 349
USING 777, 781, 782, 783, 784, 913, 989
UT 643

V-Format 897, 903, 921, 928
V-Typ 796, 797, 814, 869
VALUE 397, 647, 648, 654, 655, 656, 657, 659, 662, 663, 664, 678, 679, 680, 683, 684, 685
VALUES 685
Variable 387, 394, 395, 396, 397, 399, 401, 402, 405, 406, 407, 408, 409, 410, 411, 413, 416, 418, 419, 420, 422, 427, 430, 431, 433, 434, 439, 440, 443, 452, 453, 455, 459, 460, 461, 469, 477, 478, 481, 482, 484, 485, 486, 488, 490, 499, 500, 501, 503, 504, 511, 513, 519, 520, 521, 530, 533, 535, 536, 537, 545, 547, 548, 549, 552, 557, 559, 560, 561, 562, 571, 572, 573, 574, 575, 578, 582, 583, 588, 589, 590, 598, 599, 974
Variable Dimensionsgrenze 681, 682
Variable Länge 467, 489, 531, 532, 533, 645, 840, 872, 903, 925, 988
Variable, arithmetische 398, 443, 499
Variable, basisbezogene 476, 477, 478, 527, 531, 533, 534
Variable, beigeordnete 594
Variable, definierte 519, 521
Variable, doppelt genaue 548
Variable, duale 452
Variable, ganzzahlige 547, 548, 549, 579, 590, 591, 592, 596
Variable, indizierte 437, 451, 581
Variable, komplexe 482
Variable, nichtindizierte 478, 583, 587, 588, 589
Variable, numerische 521
Variable, reelle 547, 548, 560, 576, 579, 598
Variable, skalare 451, 455, 457, 482, 509
Variablenliste 559
Variablenname 394, 414, 435, 557, 598
Varianz 30
VARYING 523, 531, 538
Vektor 435, 450, 456, 514, 679
Veränderliche Adresse 768
Verarbeitung 360, 367, 373, 374, 378, 386, 390, 405, 414, 428, 470, 478, 484, 487, 489, 522, 634, 649, 683, 692, 699, 845, 863, 864, 868
Verarbeitung, direkte 958, 960
Verarbeitung, sequentielle 825, 826, 827,

828, 829, 830, 868, 869
Verarbeitungsblock 971, 973
Verbindungs- und Aufbereitungszeit 869
Verbindungsinstruktionen 299
Verbundanweisung 426
Verbundbedingung 715, 716
Verbundene Bedingung 715, 716
Vereinbarung 394, 396, 397, 402, 403, 405, 407,419, 422, 423, 424, 429, 434, 438, 440, 441, 459, 464, 468, 469, 470, 476, 486, 506, 507, 509, 513, 522, 524, 534, 537, 555, 570, 571, 574, 575, 576, 581, 584, 588, 589, 593, 599, 600, 649, 674, 684, 685, 715, 776, 777, 781, 798, 803, 806, 807, 814, 848
Verfalldatum 341, 381, 383
Verfügbarkeitsstatus 826
Vergleich 409, 410, 411, 412, 440, 444, 712
Vergleich, algebraischer 409
Vergleich, nichtnumerischer 712
Vergleich, numerischer 712
Vergleichsinstruktionen 71, 86, 219, 224, 227, 237, 250, 258
Vergleichsausdruck 477, 529
Vergleichsbedingung 711, 714, 716
Vergleichsinstruktion 545
Vergleichsoperation 716
Vergleichsoperator 410, 411, 716
Vergleichsprüfung 715
VERIFY 468
Verkehrsart 879
Verkettung 388, 961
Verknüpfte Datei 833, 835
Verschachtelung von IF-Statements 711
Verschiebliche Programmbibliothek 824
Verschiebliche Programme, Bibliothek 869
Verschieblicher Ausdruck 780, 781, 795, 796, 808
Verschieblicher komplexer Ausdruck 781, 795
Verschiebliches Statement 330
Verschiebliches Symbol 796
Verschiebung 525, 526, 808
Verschiebungsinstruktion 784
Verschiebungsverzeichnis 328
Verschlüsselung 2, 14, 47, 391, 409, 487, 570
Versionsnummer 342
Verteiler 971
Verteilung (nach Gauss) 31
Verteilungsstruktur 972
Verwaltungskraft 970
Verwendungsklausel 646, 676, 683, 685
Verzeichnis 361
Verzweigung 387, 403, 409, 440, 495, 509, 554, 558, 585, 586, 588, 636, 805
Verzweigung, absolute 554
Verzweigung, unbedingte 554, 787
Verzweigungen 69, 70, 73, 283
Verzweigungsadresse 782, 795, 855
Verzweigungsanweisung 584
Verzweigungsbedingungen 69, 70
Verzweigungsinstruktion 545
Verzweigungsinstruktionen 69, 70, 230, 231, 232, 233
Verzweigungsstruktur 972
Vielfache Formatangabe 573
Vielfachkonstante 792, 793
VIO 351
Virtual Input/Output 351
Virtual Sequential Acces Method 375
Virtuell sequentiell organisierte Datenmenge 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384
Virtuell sequentielle Zugriffsmethode 375
Virtuelle Adresse 267, 268, 270, 273
Virtuelle EA 351, 355
Virtuelle Partition 334, 340
Virtuelle Speichertechnik 325
Virtueller Hauptspeicher 264, 266
Virtueller Speicher 266, 331, 337, 344, 349, 351, 353, 354, 374, 375, 888, 982
Virtueller Speicherplatz 339
Virtueller Speicherumfang 337
Virtuelles Gerät 351
VIS 329
Voll gekennzeichneter Name 458
Vollendung 367
Vollwort 546, 646, 974
VOLUMES 379, 380, 381, 382
Vorgesehene Konstante 639, 683
Vorsatz 312, 823, 838, 839, 841, 878, 892
Vorschaltregister 279
Vorschub 421, 422, 426, 571, 691, 697, 703, 826, 827, 836, 892
Vorschubsteuerung 175, 176, 571, 572
Vorübersetzer 537
Vorübersetzerstatement 536, 537
Vorübersetzerzeit 536, 537
Vorübersetzung 537
Vorwärtslesen 839, 920, 921

Vorwärtssetzen 827
Vorwärtszeiger 535
Vorzeichen 3, 18, 67, 81, 82, 83, 85, 224, 236, 276, 399, 417, 460, 461, 560, 562, 646, 647, 657, 658, 661, 663, 713, 794
Vorzeichenbedingung 714
Vorzeichenüberlochung 460
VSAM 331, 338, 339, 342, 343, 347, 348, 375, 376, 378, 379, 465, 466, 490, 491, 894, 902, 948, 949, 952, 955, 960, 962, 963, 964, 965
VSAM-Hauptkatalog 384
VSAM-ISAM 376
VSAM-Katalog 331, 375, 380, 383, 384, 950, 955, 962, 963
VSO 376, 380
VS1 348, 349, 353, 354, 355, 937
VS2 348, 349, 353, 354, 355, 356, 358, 904, 937
VTOC 317, 318, 348

Währungszeichen 460, 462, 547, 637, 661, 663
Wahlfreie Organisationsform 368, 369, 374, 375
Wahlmöglichkeit 825
Wahlweises Drucken 802
Wahr 409, 444, 450, 711, 713
WAIT 916, 918, 921, 924, 925, 928, 931
WAITF 860, 861, 867
Warten 367, 853
Warteschlange 327, 348, 349
Wartestatus 93, 111, 825
Wartezustand 367, 415
Wartungsprogrammierer 970
Wechselbandgerät 333
Wert, erhöht genau 561
Wertbereich 24, 83, 89, 216, 329, 358, 546, 548, 549, 588, 645, 685, 707, 795, 804
Werteliste 438
Wertetafel 24
Wertevorrat 394, 395, 396, 441, 548, 664, 681, 974, 975
WHEN 682
WHILE 450, 451, 452
Wiederauffinden 864
Wiederbetretbar 973
Wiederholungsfaktor 391, 418, 438, 439, 454, 461, 575, 646, 647, 790, 791, 798
Wiederholungsspezifikation 453
Wiederverwendbar 973
Winkelfunktion 39, 40
Wirksamer Status 428, 429, 528
WITH 698, 699
WORDS 690
WORKING-STORAGE 644, 648
Wort 53, 61, 216, 249, 524, 550, 637, 639, 660
Wort, nichtreserviertes 648
Wort, optionales 640
Wort, reserviertes 389, 394, 638, 674, 675, 710
Wortadresse 53
Wortgrenze 523, 526, 798
Wortlänge 53
WRITE 469, 470, 478, 484, 485, 486, 488, 489, 490, 522, 557, 558, 570, 572, 574, 575, 591, 592, 593, 594, 595, 596, 598, 649, 650, 697, 698, 699, 852, 853, 854, 855, 857, 858, 859, 860, 861, 862, 866, 867, 868, 893, 894, 897, 918, 919, 920, 922, 923, 924, 925, 926, 927, 928, 929, 930
WRITE-ONLY 693, 694
WRITE-VERIFY 693, 694
WTO 937
WTP 937, 938
Wurzelphase 340
Wurzelsegment 512
WXTRN 814

X-Typ 791, 792, 793, 798, 799, 810

Y-Typ 796, 797

Z-Typ 793, 798
Zähler 707
Zählmaschine 1
Zahl, komplexe 395, 418, 433
Zahlengerade 18
Zahlensystem 20
Zahlzeichen 2, 21, 388
ZDIV 431
ZE 99, 105, 106, 110, 111, 112, 122, 124, 237, 278, 279, 355, 357
ZE-Zeitgeber 278
Zehnerkomplement 27
Zeichen 336, 340, 353, 388, 389, 390, 391, 395, 404, 405, 406, 407, 410, 413, 414, 415, 416, 419, 427, 433, 438, 439, 458, 459, 460, 463, 471, 476, 487, 495, 505, 530, 636, 637, 638, 639, 641, 646, 656, 658, 661,

662, 663, 666, 675, 697, 706, 778, 779
Zeichen, alphabetisches 388, 390, 394, 547, 637
Zeichen, alphamerisches 388
Zeichen, alphanumerisches 388, 389, 570, 637
Zeichen, nichtnumerisches 427
Zeichen, numerisches 637
Zeichendichte 182, 183, 192, 305
Zeichendirektwert 779
Zeichenfolge 358, 370, 397, 416, 420, 460, 462, 481, 571, 576, 577, 638, 802
Zeichenform 383, 481, 646
Zeichenkette 324, 390, 391, 395, 406, 409, 410, 414, 417, 418, 427, 434, 439, 442, 443, 455, 458, 459, 467, 470, 481, 483, 486, 487, 489, 505, 510, 514, 521, 530, 696, 937, 986, 987
Zeichenkettenabbildung 459
Zeichenkettendaten 409, 459
Zeichenkettenkonstante 439, 459
Zeichenkettentyp 441, 486
Zeichenkettenvariable 395, 418, 460, 480, 486, 489, 498, 521, 532, 537
Zeichenkettenvergleich 409
Zeichenklassendaten 521
Zeichenkombination 388, 389
Zeichenkonstante 789, 790, 791, 792
Zeichenmenge 47
Zeichensatz 46
Zeichentyp 789
Zeichenvorrat 13, 46, 47, 388, 389, 391, 547, 637
Zeiger 375, 377, 378, 476, 477, 488, 490, 533, 535
Zeiger-Kennzeichnungssymbol 477, 533
Zeigerausdruck 476
Zeigerdaten 474, 475, 476, 477
Zeigerergibtanweisung 476
Zeigerfunktion 476
Zeigerkennzeichnung 533
Zeigerkennzeichnungssymbol 477, 533
Zeigervariable 476, 477, 478, 485, 500, 533, 534, 536
Zeigerwert 476, 477
Zeile 380, 421, 422, 423, 454, 674, 675, 697, 774, 802, 804
Zeilen (von Lochkarten) 2
Zeilen, überspringen von 802
Zeilendrucker 47, 57, 102, 174, 175, 176, 177, 178, 179, 180, 330, 351, 359, 388, 405, 466, 571, 643, 691, 697, 699, 771, 827, 906
Zeilennummer 635, 706
Zeilenpositionsfehler 880
Zeilentransport 176, 421, 422, 466, 572, 696, 697, 698, 703, 826, 827, 892, 893
Zeilenverkehr 33
Zeiteinheit 940
Zeitgeber 91, 247, 248, 275, 276, 277, 178, 887
Zeitintervall 939
Zelle 368, 857, 867
Zentraleinheit 57, 106, 185, 237, 247, 248, 274, 278, 279, 281, 285, 355, 356, 851
ZERO 638, 639, 682, 712, 714
ZERODIVIDE 428, 431
ZEROES 639
ZEROS 639, 666, 667
Zieldatei 405
Zielfeld 648
Ziffer 2, 21, 46, 216, 336, 340, 352, 388, 392, 393, 395, 462, 486, 636, 637, 638, 643, 646, 666
Ziffer, führende 462
Ziffer, signifikante 578
Ziffernfolge 392, 578
Ziffernlochungen 2, 14
Ziffernstelle 396, 460, 461, 462, 793
Ziffernteil 80, 793
Zonenlochungen 2, 14, 81
Zonenteil 46, 48, 80, 81, 293, 793
Zugang 373, 374, 375
Zugangsfolge 375, 377, 378, 379, 381, 790, 957, 959, 962
Zugriff 367, 372, 374, 465, 959, 962
Zugriff, asynchroner 954
Zugriff, direkter 368, 377, 486, 488, 595, 864, 867, 954
Zugriff, erweiterter 367
Zugriff, fortlaufender 367, 864
Zugriff, gepufferter 367
Zugriff, sequentieller 367, 595, 596, 864, 954
Zugriff, synchroner 954
Zugriff, ungepufferter 367
Zugriffsarm 196
Zugriffsart 367
Zugriffsmechanismus 100, 196, 368, 849
Zugriffsmethode 373, 375, 464, 466, 485, 486, 893, 894, 897, 921, 924
Zugriffsmethode, direkte 375, 464, 465, 466, 486, 488, 490, 894

Zugriffsmethode, erweiterte 898, 919
Zugriffsmethode, logisch fortlaufende 894
Zugriffsmethode, sequentielle 367, 375, 464, 478, 479, 484, 485, 486, 488, 489, 490, 894
Zugriffsmethode, virtuell sequentielle 375
Zugriffsmöglichkeit, direkte 326, 348, 351, 360, 368, 464, 485, 593, 595, 643, 645, 699
Zugriffszeit 196
Zuordnung 354, 361, 452, 500, 514, 520, 523, 534
Zuordnung, sekundäre 380
Zuordnungszähler 781, 807, 808, 809, 810, 811, 812
Zurückschreiben 484, 925
Zurücksetzen 593, 827
Zurücksetzen, logisches 593
Zurückspulen 592, 827, 840, 841, 852, 982
Zusammenfassung von Attributen 432
Zusammengesetzter Ausdruck 780
Zuweisung 332, 333, 334, 335, 344, 345, 360, 406, 438, 439, 440, 458, 461, 477, 481, 501, 529, 530, 532, 598, 599, 695
Zuweisung, permanente 334
Zuweisung, temporäre 333
Zuweisungsklausel 642
Zuweisungszeichen 388, 401
Zwei-Kanal-Schalter 208
Zweidimensionaler Bereich 679
Zweierkomplement 27, 28, 61, 65, 72, 217
Zweig 529, 710
Zweizeilendruck 897
Zweizeilendrucker 836
Zwischenergebnis 671, 672, 673
Zwischenraumzeichen 2, 388, 637
Zylinder 196, 200, 203, 319, 343, 344, 360, 361, 368, 369, 370, 371, 372, 373, 378, 379, 382, 468, 489, 857, 867, 878
Zylinderende 861, 862, 880
Zylindergrenze 382
Zylinderindex 373, 375, 379, 897
Zylinderindexbereich 867
Zylinderinhaltsverzeichnis 372, 373, 375, 489, 864
Zylindernummer 368
Zylinderüberlaufbereich 373
Zylinderüberlaufsteuersatz 370, 371

Neuerscheinungen 1976

Leonhardt, Grundlagen der Digitaltechnik

Eine systematische Einführung. Etwa 180 Seiten, 4oo Bilder. 1975. Alkorphaneinband etwa 28.-- DM.

Das Buch bringt eine systematische Einführung in die Methoden und Probleme der Digitaltechnik. Um das Verständnis für digitale Schaltungen zu fördern, wird zunächst in vielen Fällen eine klassische Schaltung mit Schützen, Relais und Schaltern untersucht und dann gezeigt, wie das gleiche Problem kontaktlos gelöst werden kann.

Brinch Hansen, Betriebssysteme

Etwa 360 Seiten, Diagramme und Programmabläufe. 1976.

Die Darstellung, die sich durch besondere Klarheit auszeichnet, vermittelt Programmierern in der Praxis und Studierenden der Informatik das Wissen zum Verständnis der Funktionsweise von Betriebssystemen und bespricht die Softwaretechniken, die zur Konstruktion von Betriebssystemen erforderlich sind.

Itzinger, Maschinelle Intelligenz

Eine Einführung. 1976. Kartoniert.

Als Einführung gedacht, umfaßt das Werk sowohl die Methodik als auch Anwendungsgebiete von der mechanischen Beweisführung, von Theoremen und von heuristischen Suchmethoden. Einer Diskussion des Resolutionsprinzips von J.A. Robinson folgt eine Besprechung der wichtigsten weiteren Beweisführungsstrategien sowie eine Darstellung von Anwendungsmöglichkeiten im Bereich der Programmsynthese, der Programmverifikation und des problem-solving.

Gernert, Benutzernahe Programmiersprachen

80 Seiten. 1976. Kartoniert etwa 18.-- DM.

Diese Darstellung, die bewußt aus der Sicht des Anwenders geschrieben wurde, befaßt sich kritisch mit den Eigenschaften problemorientierter Programmiersprachen, soweit diese in der Praxis eingesetzt werden. Der Verfasser verwertet Erfahrungen aus abgehaltenen Lehrgängen und einschlägige Ergebnisse aus der Lern- und Denkpsychologie

Das Programmier-Handbuch der IBM /370

Dieses grundlegende Werk ist Handbuch, Nachschlagewerk und Lehrbuch zugleich über das gesamte Basiswissen, die Programmierung, die Programmiersprachen und den Aufbau des Systems IBM/370 mit allen seinen Modellen. Für die deutsche Ausgabe wurde das in Amerika so außerordentlich erfolgreiche Werk neu bearbeitet und auf den letzten Stand gebracht. Die Abbildungen und Zeichnungen wurden neu gestaltet. Der "Germain" behandelt sowohl kaufmännische als auch mathematisch-technische Anwendungen. Die Aufgabenstellungen und Lösungen sind beiden Problemkreisen entnommen.

Die IBM Nachrichten schrieben über den "Germain":
« . . . das gesamte für die Praxis bedeutsame Grundwissen zur Datenverarbeitung in einem Buch untergebracht. Es ist nicht nur ein Lehrbuch für Anfänger, sondern auch ein Handbuch und Nachschlagewerk für erfahrene Programmierer.»

Carl Hanser Verlag
München Wien

ISBN 3-446-